Hardware Acceleration for Homomorphic Encryption

Joël Cathebras

université PARIS-SACLAY

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

NNT : 2018SACLS576

Thèse de doctorat

Hardware Acceleration for Homomorphic Encryption

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

Ecole doctorale n°580 sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Palaiseau, le 17 décembre 2018, par

JOËL CATHÉBRAS

Composition du Jury :

Lionel Torres Professeur, Université de Montpellier, LIRMM	Président
Lilian Bossuet Professeur, Université Jean-Monnet de Saint-Etienne, Laboratoire Hubert Curien	Rapporteur
Arnaud Tisserand Directeur de recherche CNRS, Lab-STICC	Rapporteur
Caroline Fontaine Chargée de recherche CNRS, CNRS & ENS Cachan, LSV	Examinateur
Mariya Georgieva Associate researcher, EPFL & Inpher, Inc.	Examinateur
Renaud Sirdey Directeur de recherche, Commissariat à l'Énergie Atomique (CEA), List	Directeur de thèse
Alexandre Carbon Ingénieur Chercheur, Commissariat à l'Énergie Atomique (CEA), List	Encadrant
Nicolas Ventroux Chef de laboratoire, Commissariat à l'Énergie Atomique (CEA), List	Co-Encadrant
Peter Milder Associate professor, Stony Brook University, Department of ECE	Invité

Contents

Remerciements

Maintenant que ce travail de thèse est terminé, je mesure ce qu'il doit à la présence de nombreux acteurs, et puisque j'en ai la possibilité, j'aimerais qu'ils trouvent ici l'expression de ma gratitude.

Celle-ci va en premier lieu à mon employeur et à mon unité de rattachement, sans qui je n'aurais pas effectué ce travail, le Commissariat à l'Énergie Atomique (CEA) et le Département Architecture Conception et Logiciels Embarqués (DACLE) du Laboratoire d'Intégration des Systèmes et Technologies (LIST), qui ont financé mes travaux et qui ont mis à ma disposition le matériel pour les mener à bien.

Merci à Arnaud Tisserand d'avoir jugé mon travail avec bienveillance et intérêt lors de mon jury à mi-parcours, puis d'avoir accepté d'en être rapporteur en vue de sa soutenance. J'ai été honoré de la confiance qu'il m'a manifesté en me donnant l'opportunité de présenter mes travaux au séminaire sécurité des systèmes électroniques embarqués de Rennes.

J'aimerais également remercier Lilian Bossuet d'avoir accepté d'être rapporteur de ce travail, de l'avoir examiné avec bienveillance, et de m'avoir fait l'honneur de le trouver pertinent.

Je souhaite remercier Lionel Torres, que je sais par ailleurs très occupé, de m'avoir fait l'amitié d'accepter de participer à ce jury et de le présider. Merci également à Caroline Fontaine et Mariya Georgieva d'avoir apporté dans ce jury une expertise cryptographique supplémentaire et d'avoir jugé ce travail digne d'intérêt.

Enfin, je souhaite exprimer un remerciement tout particulier à Peter Milder pour la collaboration que nous avons eu pendant trois ans et pour sa participation en vidéoconférence à mon jury de thèse. J'espère que nous aurons l'occasion de travailler de nouveau ensemble.

Ma reconnaissance va également à mon encadrement pour sa présence tout au long de ces trois ans. Je veux en premier lieu remercier mon directeur de thèse Renaud Sirdey. Merci Renaud pour nos nombreux échanges qui ont grandement contribué à mon épanouissement dans mon sujet de recherche, et merci tout particulièrement pour le calme avec lequel tu as tempéré mon sur-perfectionnisme et mon manque de confiance en moi.

Je souhaite également remercier Nicolas Ventroux, chef du LCE, et Alexandre Carbon pour m'avoir accueilli dans leur équipe et confié ce sujet de thèse. Merci pour votre présence et votre soutien, pour vos relectures attentives, et pour m'avoir encouragé à valoriser la forme tout autant que le fond.

J'aimerais remercier toutes les personnes avec qui j'ai partagé, de près ou de loin, mon quotidien professionnel pendant ces trois ans. Beaucoup restent anonymes dans ces lignes mais ont contribué à ma bonne intégration dans ce cadre de travail. Certaines y ont participé plus directement et j'aimerai ici les remercier : Marie-Isabelle, Colette, Sandra et Murièle pour

leur précieux soutien, qu'il soit administratif ou logistique ; Mickaël pour sa disponibilité pour résoudre les petits problèmes informatiques du quotidien ; Renaud pour toute l'expérience d'intégration d'IP qu'il m'a partagé. Je désire aussi exprimer ma gratitude à Khaliff pour son écoute et son aide précieuses en conception FPGA (son domaine d'*expertise*). Merci également à Thierry pour son intérêt et son expérience scientifiques qui ont enrichi à plusieurs reprises mes réflexions, et à Jean-Marc pour ses bon-plans et son aide bienvenue plus d'une fois.

Je n'oublie pas mes collègues doctorants : compagnons de voyages, de petits ou de grands pots, de profondes ou de légères discussions, de galères inattendues, d'aide mutuelle et de rédactions endiablées. Merci Julien, Vincent, Johannes, Jason, Gabriel et Guillaume, c'est un plaisir d'avoir appris à vous connaitre petit à petit. Les gentils stagiaires sont aussi à l'honneur ici : merci Aurore, Fabio et Thibault pour votre présence colorée à la dynamique sympathique. Une mention particulière à Aurore et Jason pour vos relectures efficaces et bienvenues. Ma gratitude s'exprime également à tous les collègues avec qui j'ai partagé un bureau et qui ont su apprécier les expressions singulières de ma joie et de ma fatigue.

J'aimerais remercier enfin les personnes impliquées d'une autre manière dans ces trois années de ma vie. En premier lieu mes parents et mes frères qui ont vécu, sans (trop) rechigner, les montagnes russes de mon moral. Merci Paul, Charles et Raphaël d'avoir composé avec mes disponibilités éparses et de me faire la joie de votre amitié depuis si longtemps. Merci Ludivine pour ta compagnie et les formules voyage dont nous avons le secret. Merci Simon et Robin pour vos relectures aux temps forts de rédaction.

Enfin, je souhaite également remercier toutes les personnes qui ont pensé à moi pendant ces cinq derniers mois. Je ne saurais vous citer tous, mais sachez que vos messages et/ou votre présence à ma soutenance me sont allés droit au cœur.

Pour conclure, j'aimerais remercier deux personnes qui sont des jalons singuliers de ma vie : merci François-Xavier pour ton amitié et nos échanges vivifiants; merci Marie-Émeline pour ta présence offerte simplement et pour tout ce qui en naît.

Résumé en français

Introduction

Les technologies numériques se sont très vite imposées comme essentielles au fonctionnement des sociétés actuelles, et cette tendance ne semble pas sur le point de s'inverser. L'accès croissant des pays en développement à ces technologies, la forte proportion d'utilisateurs au sein des jeunes générations, ainsi que le confort qu'elles apportent au quotidien, sont autant de raisons de croire qu'elles ne disparaîtront pas dans un avenir proche. La diffusion des technologies du numérique pose la question de la confidentialité des données et conduit à la recherche de réponses techniques aidant à protéger la vie privée (voir, par exemple, le concept de Privacy-by-Design [1]). Cette thèse entend contribuer à ces enjeux, en considérant notamment le domaine de la cryptologie.

Science du secret au sens étymologique du terme, l'objet d'étude de la cryptologie est plus vaste que la simple confidentialité de l'information. Son but est de proposer des outils et de bonnes pratiques pour garantir l'intégrité, l'authenticité et la confidentialité des informations à protéger. À cette fin, elle est divisée en deux disciplines : la cryptographie et la cryptanalyse. La cryptographie est consacrée à la protection de l'information, et la cryptanalyse consiste à retrouver l'information ainsi protégée. Cette opposition permanente est une source de perfectionnement des outils cryptographiques [2, 3]. Dans cette thèse, notre intérêt s'oriente davantage vers la cryptographie, mais il faut garder à l'esprit que ces deux disciplines ne doivent pas être considérées indépendamment.

Face à l'augmentation des coûts de maintenance des infrastructures informatiques, le marché des services informatiques centralisés connaît une forte croissance. Par exemple, l'essor de la virtualisation et des réseaux à très haut débit permet à des entités spécialisées de fournir des espaces de stockage et de la puissance de calcul à d'autres entités dont le numérique n'est pas le cœur de métier. La question de l'externalisation de données confidentielles, que ce soit pour leur stockage et/ou leur traitement, est donc pertinente. La cryptographie classique (symétrique et asymétrique) ne permet que de sécuriser l'échange et le stockage des informations. Mais il est également nécessaire de garantir la confidentialité des données pendant les traitements.

En 1978, Rivest, Adleman et Dertouzos introduisirent la notion de *cryptographie homomorphe* qui permettrait de tels services. Le principe est d'avoir un processus de chiffrement qui préserve une structure entre le domaine chiffré et le domaine clair, permettant ainsi de définir des opérations dans le domaine chiffré à effet équivalent aux opérations usuelles sur les clairs. Ce n'est qu'à la fin des années 2000 que la cryptographie homomorphe a véritablement émergé avec une première solution théorique. Depuis, de nombreux travaux de recherche ont permis de se rapprocher d'une mise en application de ce nouveau type de cryptographie. D'une

part, ces travaux simplifient les constructions mathématiques sous-jacentes aux schémas de chiffrement homomorphe, et d'autre part, ils explorent des optimisations algorithmiques pour profiter d'implémentations plus efficaces.

Néanmoins, ce paradigme cryptographique semble induire intrinsèquement des complexités mémoire et calculatoire importantes. Par exemple, un chiffré homomorphe est environ 10^5 fois plus grand que la donnée qu'il chiffre, et une opération dans le domaine chiffré est environ 10^6 fois plus lourde que son équivalente claire. En conséquence, les acteurs impliqués dans la recherche sur la mise en application de cette nouvelle cryptographie ont besoin d'optimisations supplémentaires telles que l'accélération matérielle.

Cette thèse

La cryptographie homomorphe est un domaine encore en éclosion. Bien que de nombreuses applications soient à l'étude, peu ont le mérite d'être pratiquement explorées, et encore moins sont déjà réalisables à grande échelle. Néanmoins, la recherche progresse rapidement vers son utilisation en dehors des laboratoires, et nous espérons que certaines des contributions associées à cette thèse pourront aider ce développement.

La problématique. L'objectif principal de la recherche sur la cryptographie homomorphe est de définir des schémas de chiffrement pour lesquels il y a un équivalent dans le domaine chiffré pour tout traitement applicable sur des données. Cet objectif est appelé *chiffrement entièrement homomorphe* (FHE : *Fully Homomorphic Encryption*). Le fait qu'il se soit écoulé près de trente ans entre l'introduction du concept et la première construction théorique d'un schéma n'est pas un hasard. En effet, deux problèmes théoriques se recoupent.

Le premier est la difficulté de trouver des expressions de problèmes mathématiques, assurant la sécurité des schémas, qui permettent de construire des opérations équivalentes. Le second est la présence nécessaire, aussi pour des questions de sécurité, d'un bruit de chiffrement. L'amplitude du bruit dans les chiffrés augmente avec les opérations dans le domaine chiffré. Au-delà d'un certain seuil, le processus de déchiffrement cesse de fonctionner.

Avec les recherches qui ont suivi la première construction théorique d'un schéma FHE, certaines solutions intermédiaires ont rapproché la cryptographie homomorphe d'un usage concret. De même, la recherche sur l'accélération matérielle de ces solutions intermédiaires doit faire face à deux difficultés principales. La première est la variété des schémas proposés : en moins de dix ans, quatre générations de schémas ont déjà été proposées. Ainsi, les avantages et inconvénients des schémas n'ont pas pu être pleinement et objectivement comparés sur une période de temps si courte. Il en résulte donc un positionnement plus qualitatif dans le choix des schémas à considérer pour une accélération matérielle. La deuxième difficulté est que le paramétrage de ces schémas est fortement dépendant de l'application à réaliser dans le domaine chiffré. Cela est dû principalement à la croissance du bruit de chiffrement, qui exige que les paramètres soient correctement dimensionnés. Ainsi la stratégie d'accélération doit être suffisamment flexible pour rester pertinente avec l'accroissement des paramètres.

Contributions scientifiques. Au cours de ce travail, nous avons surtout abordé la question de la dynamique des paramètres. La littérature a proposé les éléments de base sur lesquels reposent nos contributions, à savoir d'une part l'utilisation du *système modulaire de représentation* (RNS : *Residue Number System*) pour s'attaquer à la complexité apportée par l'arithmétique des grands nombres entiers, et d'autre part l'utilisation des transformées de

Fourier sur corps finis (NTT : *Number Theoretical Transform*) pour réduire la complexité algorithmique asymptotique de la multiplication polynomiale. Sur cette base, nous soutenons que l'association du RNS et de la NTT permet de définir des architectures hybrides de calcul, qui promettent une accélération significative des traitements dans le domaine chiffré. Cette thèse résulte d'une analyse approfondie du chiffrement de Fan et Vercauteren (FV), ainsi que des travaux sur la résolution des principales difficultés à combiner la représentation RNS et la multiplication de polynôme par NTT pour des grands paramètres.

Nos principales contributions sont les suivantes. Tout d'abord, nous avons analysé l'approche couplée RNS/NTT et théoriquement validé sa faisabilité jusqu'à de très grands ensembles de paramètres. Deuxièmement, nous avons exploré l'accélération GPU de certains algorithmes spécifiques au RNS, grâce à l'utilisation et à la poursuite des travaux de Bajard et al. [4] et Halevi et al. [5] pour adapter complètement FV au RNS. Troisièmement, nous avons conçu et mis en œuvre un accélérateur orienté flux de données pour des multiplications de polynômes résiduels efficaces et étudié sa capacité à passer à l'échelle.

Cette dernière contribution fait l'objet d'une attention particulière afin de surmonter certains problèmes de conception qui limitaient auparavant la faisabilité de l'approche couplée RNS/NTT pour des grands paramètres. En particulier, nous proposons une solution pour précalculer localement les valeurs nécessaires aux opérations de NTT, sans impact sur les performances. Ceci est obtenu grâce à deux contributions : la conception d'un circuit de NTT changeant ses corps finis de définition à la volée, et la conception d'un générateur de facteur de rotation pour les NTT sur les différents corps finis. Enfin, pour répondre à la question de la grande variation des paramétrages en fonction des différentes applications du chiffrement homomorphe, la génération automatique des descriptions HDL (*Hardware Description Language*) de nos circuits a été explorée.

État de l'art et positionnement

Les chiffrements homomorphes sont capables d'effectuer des opérations sur des données chiffrées sans les déchiffrer au préalable. Cette capacité vient du choix d'un homomorphisme comme fonction de déchiffrement d'un schéma. La définition d'un homomorphisme met directement en évidence la caractéristique désirée : il s'agit d'une correspondance conservant une certaine organisation entre deux structures algébriques du même type (groupes, anneaux, corps...). En considérant l'espace des clairs et l'espace des chiffrés comme des structures algébriques, un homomorphisme consiste alors à avoir une équivalence entre certaines opérations dans les domaines chiffré et clair.

On qualifie de *partiellement homomorphe* (*Partial Homomorphic Encryption* : PHE) un chiffrement dont l'ensemble des fonctions évaluables est restreint à un sous-ensemble des fonctions internes sur l'ensemble des clairs. Par exemple, si la fonction de déchiffrement est un homomorphisme de groupe additif (resp. groupe multiplicatif), alors seules les fonctions composées d'additions (resp. de multiplications) auront un équivalent dans l'ensemble des chiffrés. On parlera de chiffrement *presque homomorphe* (*Somewhat Homomorphic Encryption* : SHE) lorsque la fonction de déchiffrement est un homomorphisme d'anneaux, c'est à dire qu'il leur est possible d'évaluer à la fois des additions et des multiplications dans le domaine chiffré, mais que ces chiffrement sont limités en opération dans le domaine chiffré par l'accroissement du bruit de chiffrement. Enfin, on parlera de chiffrement *complètement homomorphe* (Fully Homomorphic Encryption : FHE) si toutes les fonctions internes de l'ensemble des clairs appartiennent à l'ensemble des fonctions évaluables dans le domaine chiffré.

Analyse de l'état de l'art des chiffrements homomorphes. L'effervescence de la recherche autour de la cryptographie homomorphe peut être déroutante pour qui s'intéresse à des considérations pratiques telles que la sécurité et la performance. La première génération résulte de la première construction de schéma complètement homomorphe proposé par Gentry en 2009 avec l'introduction d'une opération de réamorçage (*bootstrapping*) très coûteuse. La deuxième améliore la gestion du bruit des schémas en évitant de passer par l'opération de réamorçage, et introduit ainsi la notion de *chiffrement homomorphe par niveau* (L-FHE : *Leveled Fully Homomorphic Encryption*). La troisième génération est une simplification conceptuelle de la deuxième génération et propose des schémas homomorphes par niveau structurellement plus simples. Enfin, la quatrième génération réintroduit une opération de réamorçage plus performante. Cette fois, c'est chaque opération sur les chiffrés qui réamorce le bruit.

Lorsque l'on s'intéresse à la maturité des schémas, il semble que ceux qui sont construits sur les variantes du problème mathématique LWE (et en particulier ceux basés sur RLWE) soient actuellement les plus équilibrés en termes de sécurité et d'efficacité. En effet, les améliorations de la deuxième génération, puis des troisième et quatrième générations, dépendaient principalement de l'apparition du problème de l'*apprentissage avec erreurs* (LWE : *Learning With Errors*). Les schémas basés sur LWE manipulent des matrices et/ou des vecteurs d'entiers modulaires. La variante RLWE (Ring-LWE) manipule des matrices et/ou des vecteurs de polynômes modulaires. La quatrième génération construit également ses schémas sur des variantes du problèmes LWE. Cependant, son schéma le plus prometteur (TFHE [6]) a choisi une structure algébrique légèrement différente des autres pour améliorer les performances de ses primitives. Il manipule des polynômes à coefficients réels modulo un. Par conséquent, ce schéma n'est pas confronté aux mêmes problématiques d'implémentation et nécessite des approches différentes des schémas basés sur RLWE.

Au moment de notre positionnement, la littérature nous a amené à considérer que les schémas construits sur RLWE sont ceux qui se rapprochent le plus d'une utilisation concrète. En particulier, les schémas FV [7] $2^{\text{ème}}$ génération) et SHIELD [8] $3^{\text{ème}}$ génération) sont bien acceptés par la communauté. Notre étude de l'état de l'art de l'implémentation du chiffrement homomorphe s'est donc concentrée sur les problèmes propres à ce type de schémas.

Analyse des stratégies d'implémentation existantes. Le choix de considérer les implémentations matérielles pour les chiffrements homomorphes était un parti pris de cette thèse. Ceci se justifie par les limitations en performance que l'on expérimente avec des implémentations purement logicielles. Ainsi, notre étude s'est focalisée sur les problématiques d'implémentation matérielle et l'état de l'art associé.

Le problème principal concerne la taille importante des paramètres des schémas RLWE, nécessaire pour des raisons de sécurité et d'exactitude. Ce problème se complique en considérant en plus la dynamique des paramètres due aux contraintes applicatives. Pour en avoir une petite idée, le degré des polynômes manipulés peut atteindre plusieurs dizaines de milliers, et la taille de leurs coefficients peut atteindre plusieurs centaines de bit. Par conséquent, les complexités mémoire et calculatoire des opérations sous-jacentes sont particulièrement importantes. Ces opérations sont des multiplications de polynômes, des réductions modulaires de polynômes, des additions de polynômes, des opérations de mise à l'échelle et d'arrondi, et des réductions modulaires.

Notre analyse de la littérature nous a fait constater la difficulté de proposer une approche cohérente qui puisse englober toutes les dimensions de la problématique d'implémentation.

Par exemple, les approches de multiplication de polynômes les plus flexibles permettent, au détriment de complexités asymptotiques moins intéressantes, de simplifier les opérations de mise à l'échelle et de réduction modulaire. Ces approches ne trouvent leur justification que jusqu'à une certaine taille de paramètres. Au contraire, les approches de multiplication de polynômes basées sur la transformée de Fourier sur corps fini (NTT : *Number Theoretical Transform*), qui ont à ce jour la meilleure complexité asymptotique connue, sont plus complexes à mettre en œuvre.

Une première difficulté est due au routage des coefficients pour les algorithmes de transformée de Fourier itératifs. Une deuxième difficulté concerne la gestion des facteurs de rotation des NTT. Lorsque les paramètres du schéma grossissent, l'espace mémoire requis pour ces facteurs (resp. le coût de communication pour les acheminer) augmente fortement le coût matériel (resp. la bande-passante nécessaire). Une troisième difficulté concerne la gestion des très grands coefficients. Sur cette question, l'approche la plus prometteuse semble être l'utilisation d'une représentation non-positionnelle des nombres appelée RNS (*Residue Number System*). Cependant, bien que le couplage du RNS avec les approches par NTT est théoriquement valide, le choix de la base de représentation RNS est restreint par les contraintes d'existence des NTT. Ainsi, certains choix permettant de faire de l'arithmétique modulaire à bas coût ne sont pas possibles. À cela s'ajoute la difficulté de faire efficacement les opérations de mise à l'échelle et d'arrondi en représentation RNS.

Au regard de toutes ces problématiques, il nous a semblé plus important de rechercher une approche d'accélération flexible plutôt que dédié à la performance brute. Pour cela, nous avons limité l'étude de cas à un schéma homomorphe spécifique, afin de proposer une approche qui tienne compte, autant que possible, de toutes les dimensions du problème.

Positionnement. Après avoir étudié les différentes générations de chiffrements homomorphes, nous avons concentré notre étude sur les schémas homomorphes par niveau basés sur le problème RLWE. Ce choix implique que les paramètres dimensionnants dépendent de l'application à effectuer dans le domaine chiffré. L'approche d'accélération matérielle devra donc tenir compte de la dynamique des paramètres.

Ce positionnement a été suivi d'une analyse des différentes approches d'implémentation matérielle pour l'accélération de la cryptographie homomorphe. Cette analyse pose la question de la définition d'une stratégie d'implémentation qui prenne en compte toutes les dimensions du problème de complexité mémoire et calculatoire provenant de la dynamique des paramètres. Cette stratégie devrait être suffisamment flexible pour améliorer la performance du schéma L-FHE tout en évitant les spécialisations prématurées qui ne sont pas adaptées à la dynamique des paramètres.

Afin de définir une telle stratégie à la lumière des travaux connexes, nous avons choisi de faire une étude approfondie du schéma homomorphe FV. Ce dernier est plutôt bien accepté par la communauté de recherche autour du chiffrement homomorphe. Par conséquent, la suite de notre travail s'est concentrée sur l'accélération matérielle du chiffrement FV.

Analyse du chiffrement FV

L'analyse approfondie du schéma FV en vue de son accélération matérielle nous a introduit aux problématiques concrètes du choix des paramètres. Dans un premier temps, nous avons présenté ses primitives, les structures algébriques manipulées, ainsi que les équations d'accroissement du bruit. Les paramètres dimensionnants que sont le degré des polynômes et la

taille des coefficients ont un impact majeur dans cet accroissement du bruit, surtout lors des multiplications de chiffrés. De même, ils ont un impact important sur la difficulté du problème RLWE sous-jacent assurant la sécurité du schéma. Les travaux de thèse de Guillaume Bonnoron [9] présentent une méthode de dérivation des paramètres prenant en compte tout ces aspects, ainsi que des exemples concrets de paramètres. À partir de cette vue d'ensemble, la problématique générale des schémas homomorphes par niveau est compréhensible. Afin d'évaluer des applications de taille conséquente dans le domaine chiffré, les paramètres augmentent considérablement. Ainsi, la restriction de l'éventail des paramètres n'est pas possible sans considérer des cas particuliers d'utilisation du schéma FV.

Après la présentation du schéma et de ses paramètres, l'étape suivante consiste à identifier et à quantifier les goulots d'étranglement de la performance des opérations homomorphes avec FV. Le profilage d'une application homomorphe typique a quantifié la complexité dominante des multiplications de chiffrés et des opérations de relinéarisation. Les opérations sous-jacentes de multiplication de polynômes et de mise à l'échelle sont les principales opérations limitant les performances. Sur la base de notre analyse des travaux de l'état de l'art sur ces problématiques, nous avons choisi d'explorer la faisabilité de l'approche couplée RNS/NTT pour l'accélération du crypto-calcul avec FV. Après une présentation détaillée du RNS et des multiplications de polynômes par NTT dans notre contexte, nous avons pu faire une liste de toutes les contraintes à prendre en compte pour l'utilisation conjointe de ces deux approches. L'étude de ces contraintes se termine par une validation théorique et pratique de l'approche couplée.

Une problématique particulière de l'utilisation de la représentation RNS est celle de l'opération de mise à l'échelle. Les travaux concomitants de Bajard et al. [4] et de Halevi et al. [5] ont décrit des méthodes efficaces pour en améliorer la performance dans le cas de FV. L'analyse du profilage présenté par Halevi et al. [5] affine notre propre profilage. Il oriente concrètement notre stratégie d'implémentation matérielle vers l'accélération des multiplications de polynômes résiduels, des opérations d'extension de base RNS, et des opérations de mise à l'échelle en RNS. En raison de la complexité dominante des multiplications de polynômes, nous nous sommes donc concentrés principalement sur leur accélération.

Les opérations de NTT, qui servent de briques de base aux algorithmes de multiplication de polynômes, sont assez difficiles à paralléliser en raison d'accès mémoires complexes. Cela rend les grandes NTT peu adaptées aux architectures SIMD génériques telles que les GPU. Notre choix est donc d'explorer des solutions d'accélération s'appuyant sur du matériel dédié pour ces opérations.

Étant donné les problèmes d'implémentation mis en évidence par les travaux connexes quant à la gestion des facteurs de rotation, notre stratégie consiste à générer et utiliser à la volée ces différents ensembles de facteurs. Ceci exige de s'attaquer à deux problématiques principales de conception. La première est de pouvoir générer efficacement des circuits de NTT multi-corps. Il s'agit de circuits de NTT capables d'effectuer des transformées sur différents corps finis sans impact significatif sur les performances. La seconde est la conception d'un générateur d'ensembles de facteurs de rotation pour les différents corps finis définis par la base RNS. Il faut que cette génération soit sans conséquences lourdes sur le coût du matériel et sur le débit des opérations de NTT.

Proposition de circuits de NTT multi-corps

Les problématiques de conception de circuits de NTT sont structurellement les mêmes que celles des transformées de Fourier discrète (DFT : *Discrete Fourier Transform*). Or, le domaine du traitement du signal a déjà beaucoup contribué à l'implémentation matérielle des DFT. Nous avons en particulier considéré que le projet SPIRAL [10] constitue un point de départ particulièrement intéressant pour automatiser la conception de circuit de NTT. En effet, le projet SPIRAL porte sur l'automatisation du développement logiciel et matériel pour le traitement du signal.

La perspective à moyen terme est de pouvoir concevoir rapidement des circuits de NTT selon des contraintes de coût matériel et de performance de calcul. Cela permettra à un concepteur de systèmes d'ajuster les performances de ses opérateurs de NTT en fonction des exigences applicatives et des autres éléments du système. Au cours de cette thèse, seuls les circuits en flot-de-données complètement déroulés (*fully-streaming*) ont été étudiés et adaptés en circuit de NTT multi-corps. En raison de contraintes de temps, nous nous sommes concentrés sur ce type d'architectures qui offre les débits les plus élevés. C'est en effet ce qui est principalement requis par notre contexte applicatif.

L'adaptation de SPIRAL pour la génération de NTT consiste en la génération de circuits de DFT dans lesquels l'arithmétique complexe est remplacée par de l'arithmétique modulaire et les facteurs de rotation complexes sont remplacés par les facteurs de rotation du corps fini considéré. Dans le cas d'une DFT, SPIRAL précalcule les facteurs de rotation et les stocke dans des mémoires non-programmables (ROM) à la génération du circuit. Cette approche convient à la DFT, car les facteurs de rotation ne sont pas susceptibles de changer. Cependant, notre cas applicatif nous impose de trouver un moyen de reprogrammer ces facteurs de rotation sans impact majeur sur le débit du circuit de NTT. En effet, le corps fini de définition de la NTT est différent pour chaque canal RNS (chaque élément de la base RNS), et ainsi les ensembles de facteurs de rotation sont différents.

Définition d'une NTT multi-corps. La répartition des facteurs de rotation au sein du circuit de NTT est connue par le générateur SPIRAL. Plutôt que d'implémenter les mémoires des facteurs de rotation directement dans le chemin de données du circuit de NTT, nous proposons de dissocier ces mémoires de ce dernier, afin de les rendre programmables. Nous définissons la notion de banque de facteurs de rotation, qui consiste en la concaténation de toutes les mémoires, initialement dans le chemin de données, et stockant un unique jeu de facteurs de rotation pour un corps fini spécifique. En plus des facteurs de rotation, une telle banque stocke les autres valeurs spécifiques au corps fini, comme l'élément associé de la base RNS.

Pour éviter de réduire le débit du circuit de NTT lors de la reprogrammation d'une banque de facteurs de rotation, notre solution s'appuie sur un ensemble de G banques différentes. Ces G banques vont être successivement accédées par le chemin de données et reprogrammées avec un nouvel ensemble de facteurs de rotation. Ce nombre G est lié au nombre maximum de canaux RNS pouvant être simultanément présents dans le chemin de données. Si nous notons T le débit du chemin de données (nombre de cycles d'horloge entre deux transformées) NTT et lat_{NTT} sa latence, l'architecture comporte $G = \lceil lat_{NTT}/T \rceil + 1$ banques différentes. Notre solution s'accompagne de la définition d'un module de reprogrammation des banques de facteurs qui garantie un débit de reprogrammation adapté à celui du circuit de NTT.

Analyse de notre contribution. Notre proposition de circuit NTT multi-corps implique un surcoût matériel par rapport à un circuit de NTT défini pour un unique corps fini. Ce surcoût est principalement dépendant du nombre G de banques de facteurs instanciées. Les modules contrôlant les accès et la programmation des banques ont un impact relativement faible sur la consommation de ressources matérielles. Pour les paramétrages considérés dans cette thèse, G était toujours égal à quatre. Ce surcoût qu'implique notre solution n'est pas très important par rapport aux avantages qu'il apporte dans notre contexte.

Pour mettre en évidence ces avantages, nous comparons la manipulation de notre gestion de facteurs de rotation à la volée avec une stratégie de stockage local présente dans l'état de l'art. Elle consiste à stocker tous les facteurs de rotations, pour tous les canaux RNS, dans des mémoires non-programmables (ROM) initialisées à la génération du circuit. L'avantage d'utiliser notre approche se concrétise avec l'accroissement des tailles de paramètres du schéma FV. La taille des bases RNS (nombre de corps finis différents) et le degré des polynômes (nombre de facteurs de rotation pour chaque corps fini) augmentent de manière significative. Notre traitement à la volée des ensembles de facteurs de rotation réduit de 20% à 90% les éléments de mémorisation effectivement instanciés pour fournir le même résultat. Ceci permet à notre approche d'être praticable pour des jeux de paramètres de FV significativement plus grands que l'état de l'art (profondeur multiplicative de 30 avec une sécurité de 128-bit).

Notre gestion des facteurs de rotation à la volée permet d'envisager des multiplieurs de polynômes en représentation RNS à haut débit. Néanmoins, à elle seule, cette gestion à la volée ne fait que déporter le problème du stockage de ces facteurs de rotation. Ils devront être acheminés vers le circuit de NTT multi-corps d'une manière ou d'une autre. Dans le cas où les facteurs sont stockés à l'extérieur de l'accélérateur (mémoire du processeur hôte par exemple), il faut s'attendre à une diminution des performances effectives à cause des communications plus lourdes. En effet, l'opérateur de NTT devra recevoir les ensembles de facteurs de rotation en plus des données utiles. Afin de ne pas impacter ces temps de communications, notre stratégie consiste à générer à la volée ces ensembles de facteurs.

Génération à la volée des facteurs de rotations

Après avoir proposé une approche pour générer des circuits de NTT multi-corps avec SPIRAL, la deuxième problématique que nous abordons est celle de la génération des ensembles de facteurs de rotation. Nous proposons dans cette thèse des générateurs de ces ensembles qui respectent les exigences de débit de nos circuits de NTT multi-corps. La problématique consiste donc en la génération des puissances d'éléments de différents corps finis pour laquelle notre contexte impose un débit minimal.

La première difficulté se situe dans le choix d'une méthode de génération. Cette difficulté tient à la fois de la dépendance entre les éléments d'une séquence de puissances à générer et de la latence des multiplicateurs modulaires qui produisent de nouveaux éléments. Ainsi, certaines bulles inévitables se produisent dans la génération d'un seul ensemble de facteurs de rotation. La méthode choisie impactera significativement la latence d'une génération, et le coût matérielle pour la mener à bien.

La deuxième difficulté se situe dans le respect du débit minimal requis entre les générations des différents ensembles. Du fait des inévitables bulles dans la génération d'un unique ensemble de facteurs, la génération séquentielle de ces ensembles ne permet pas d'atteindre le débit requis.

Ainsi, nous proposons des solutions qui répondent à ces difficultés. Premièrement, nous formalisons une solution pour respecter le débit requis par le circuit de NTT multi-corps entre deux ensembles de facteurs de rotation consécutifs. Cette première proposition consiste en une solution d'ordonnancement de multiples générations d'ensembles autours d'une unique ressource de calcul. Deuxièmement, nous proposons un choix de relation de récurrence pour la génération de chacun des ensembles de facteurs. Cette relation de récurrence permet de minimiser l'espace de stockage intermédiaire requis tout en minimisant la latence de chaque génération pour la quantité de ressources de calcul disponible.

Analyse de notre contribution. La génération à la volée des ensembles de facteurs de rotation permet d'alléger les communications entre une zone de stockage externe à l'accélérateur et ce dernier. Pour quantifier cet allègement des communications, nous avons comparé notre approche à une stratégie de stockage externe similaire à certains travaux de l'état-de-l'art. Les métriques de comparaisons sont les empreintes mémoires et les bandes passantes requises pour alimenter un circuit de NTT multi-corps. L'empreinte mémoire est définie ici comme la quantité de mémoire utilisée par un programme hôte pour utiliser l'accélérateur de NTT considéré.

L'empreinte mémoire des facteurs de rotation pour la stratégie de stockage externe va de 19,2 koctets à 2,6 Moctets en fonction des jeux de paramètres considérés. Notre stratégie de génération de ces facteurs requiert quand à elle seulement 2,2 koctets au maximum.

En considérant un circuit de NTT cadencé à 200MHz, la stratégie de stockage externe demande au moins 0,75 Goctets/s de bande passante entre l'espace de stockage et l'accélérateur pour les seuls facteurs de rotation. Avec notre générateur, la bande passante nécessaire pour acheminer les éléments initiaux des générations est de maximum 15 koctets/s.

De pair avec notre proposition de circuits de NTT multi-corps, notre générateur de facteurs de rotation permet la conception de circuits de multiplication de polynômes résiduels à haut débit à base de NTT. Dans le contexte de l'accélération matérielle du chiffrement FV, cela ouvre des perspectives intéressantes pour la définition d'une accélération matérielle efficace. En s'appuyant sur notre analyse générale du chiffrement FV, nous présentons une proposition d'architecture système utilisant nos blocs matériels de base pour l'accélération matérielle de la version complètement RNS de FV.

Exploration d'une accélération hybride pour l'approche couplé RNS/NTT

L'analyse du chiffrement FV menée au début de cette thèse a mis en évidence la nécessité d'accélérer à la fois les opérations de mise à l'échelle en RNS et les multiplications de polynômes résiduels. Cela s'explique par la stratégie d'accélération choisie se basant sur l'approche couplée de la représentation RNS et des multiplications de polynômes basées sur la NTT. La représentation RNS s'attaque à la complexité induite par les grands coefficients, et les approches de convolution par NTT s'attaquent à la complexité induite par le grand degré des polynômes.

Parmi les contributions de cette thèse, nous avons présenté des blocs de base pour l'accélération matérielle des opérations de multiplication de polynômes résiduels. Le choix du matériel dédié pour ces opérations est motivé par la difficulté d'exploiter le parallélisme d'une NTT sur

des architectures SIMD génériques en raison d'accès complexes aux données. Contrairement à la NTT, les fonctions spécifiques au RNS intègrent un parallélisme trivial par rapport au degré des polynômes. Ce parallélisme est facilement exploitable avec des architectures SIMD génériques comme les GPU. Ainsi, notre réflexion considère l'exploration d'une architecture hybride de calcul pour l'accélération du crypto-calcul avec FV. Les fonctions spécifiques au RNS sont accélérées par un GPU, et l'arithmétique polynomiale par un co-processeur dédié.

En plus d'étudier le gain obtenu par l'accélération hybride sur chacune des opérations accélérées, nous nous sommes également intéressés à la problématique de la communication entres les unités de calcul du système. En effet, le flot d'opérations, imposé par la structure des primitives de FV, requiert des échanges continus de données entre le GPU et le co-processeur. Quantifier ces échanges au regard des performances de chacune des unités de calcul est la première étape pour dimensionner un bus système pertinent. Au regard des quantités de données, notre première proposition d'interconnexion s'appuie sur un bus PCIe.

Analyse de notre contribution. Lors d'un stage effectué dans le cadre de cette thèse, l'accélération GPU des fonctions spécifiques au RNS a été explorée. Pour les plus petits jeux de paramètres, le temps passé en communication entre l'hôte et le GPU est plus important que les calculs eux-mêmes. Même pour l'ensemble le plus large, le temps consacré aux communications est non-négligeable. Néanmoins, même avec des temps de communication importants, les opérations spécifiques au RNS bénéficient d'une accélération significative. Pour une implémentation concrète de cette architecture hybride, cela indique que les communications entre les différentes unités de calcul devront être gérées finement pour qu'elles limitent au minimum les performances.

A partir de nos briques de base de NTT multi-corps et de générateur de facteurs de rotations, nous avons fait une proposition de co-processeur pour la multiplication de polynômes résiduels. Nos briques de base permettent la faisabilité de cet accélérateur même pour des grands jeux de paramètres. Notre accélérateur permet une accélération significative de ces multiplications de polynômes par rapport à l'implémentation de Halevi et al. [5], en étant au moins 22 fois plus rapide pendant la multiplication de chiffrés et au moins 4 fois plus rapide lors de l'opération de relinéarisation. De plus, cette accélération reste pertinente avec la croissance des jeux de paramètres, soit de 22 à 31 fois plus rapide pendant la multiplication de chiffrés et de 4 à 9 fois plus rapide pour la relinéarisation.

Pour motiver la pertinence de notre système de calcul hybride, nous avons ensuite estimé l'accélération obtenue pour le calcul d'une opération de multiplication de chiffrés et relinéarisation avec FV. Le tableau 1 présente ces projections pour différents jeux de paramètres. Ces paramètres sont caractérisés par la profondeur multiplicative accessible par les chiffrés. Le temps de communication pour les accès GPU a été pris en compte. Notre système permet d'envisager une multiplication de données chiffrées jusqu'à 14 fois plus rapide qu'une version logicielle bien optimisée (Halevi et al. [5]). Cette accélération réduit le temps de multiplication de chiffrés à 62 ms pour les paramètres permettant une capacité d'évaluation d'une profondeur multiplicative de 30, et à 21 ms pour une profondeur multiplicative de 20. Néanmoins, ces projections sont obtenues en faisant des hypothèses simplificatrices sur les communications et les entrelacements avec d'autres opérations sous-jacentes comme les additions de polynômes résiduels. Ces résultats prospectifs motivent cependant le perfectionnement de l'accélérateur dédié pour l'arithmétique polynomiale ainsi que la réalisation d'un prototype du système hybride de calcul.

Table 1: Projection de l'accélération obtenue avec notre proposition de système hybride de calcul pour la primitive FV.Mul&Relin. Les colonnes RNS regroupent les opérations spécifiques au RNS. Les colonnes RPM regroupent toutes les opérations de multiplication de polynômes requises par l'opération FV.Mul&Relin.

Parameters			Timing basis from [5]				Our hybrid system			
L	n	$\log_2 q$	**Total**	RNS	RPM	Misc.	**Total**	speedup	RNS	RPM
1	2^{11}	54	**3.6**	1.3	2.2	0.1	**0.67**	**×5.37**	0.51	0.06
5	2^{12}	108	**12.7**	3.8	8.6	2	**3.28**	**×3.87**	1.05	0.23
10	2^{13}	216	**57.6**	14.4	41.8	2.6	**7.13**	**×8.08**	3.32	1.21
20	2^{14}	432	**252**	71.3	175.3	2.2	**20.7**	**×12.17**	11.17	7.33
30	2^{15}	594	**887**	233.1	630.6	2.6	**61.21**	**×14.49**	33.14	25.47

Conclusion et perspectives

Les recherches pour du crypto-calcul efficace se poursuivent. Tout au long de cette thèse, nous avons cherché à prendre en compte la complexité de l'implémentation du chiffrement homomorphe. Cette complexité est premièrement celle du choix du chiffrement à considérer pour une accélération matérielle. Notre analyse de l'état de l'art nous a orienté vers les schémas homomorphes basés sur le problème RLWE, et en particulier le schéma FV. Deuxièmement, le choix de la stratégie d'accélération doit prendre en compte de nombreux éléments. Le plus important à notre avis est la grande variété des paramétrages qui implique la nécessité d'une stratégie d'accélération flexible. En ce sens, nous avons choisi une stratégie couplant la représentation des nombres en RNS et la multiplication de polynômes par NTT. Nos contributions montrent en particulier la possibilité d'avoir des circuits de NTT multi-corps avec une génération locale des facteurs de rotation, pour un coût matériel peu sensible aux variations des paramètres de FV.

Au cours de ce travail, nous avons ouvert la voie à la conception d'une architecture de microserveur pour le crypto-calcul avec FV. Nous avons commencé à explorer les problématiques de communication qui se poseraient dans un tel système. Quelques pistes pourraient être intéressantes à explorer dans le cas où l'interconnexion PCIe proposée dans cette thèse serait insuffisante. Par exemple, en explorant l'intégration du GPU et du matériel dédié pour l'arithmétique polynomiale plus près de la mémoire du CPU. En restant dans une approche multi-SoC, nous pourrions considérer l'interface *Coherent Accelerator Processor Interface* (CAPI) d'IBM. Pour une approche plus intégrée en un single-SoC hétérogène, nous pourrions explorer l'*UltraPath Interconnect* (UPI) d'Intel. Enfin, tant qu'à considérer la conception d'un ASIC, il serait intéressant d'examiner certaines technologies spécifiques telles que l'intégration de mémoire 3D.

La problématique de conception de système de communication à faible latence et haut débit se retrouve dans d'autres domaines, notamment ceux dont les performances de calcul dépendent davantage de la quantité de données à traiter que des calculs eux-mêmes. De ce point de vue, l'accélération du chiffrement homomorphe est similaire à l'accélération du traitement d'images et de vidéos à la volée.

Introduction

Digital technologies at the heart of our lives

It is no secret that digital technology has very quickly established itself as essential to the functioning of developed societies. And this trend does not seem to be about to reverse. A first impress of this can be obtained looking at the number of internet subscribers that increased from 400 millions to 3.2 billions between 2000 and 2015 [11]. In particular, developing countries play an important role in this growth. Considering then that the part of internet users in the younger generations is significantly larger than for older's ones [12], we can assume that the internet population will continue to grow. More generally, the ease brought by these technologies for many daily tasks gives another reason to believe that they will not disappear in the near future.

These technologies bring with them many economic, societal and cultural opportunities. For instance, specialized medical care brought by massive data collection [13]. But they can also be the source of misuses and imbalances in each of these dimensions. An example of this is the 2015 spy toy controversy [14]. This is only one example among others of abusive data collection of persons who neither have the knowledge of this collection, nor the ability to protect themselves from it. Many of these issues are being addressed by the political sphere and by national and international entities (for example, the 2018 revision of the french law *Informatique et Libertés* [15]. Addressing the moral question of the proper use of these technologies results in legal delimitation expressing the rights and responsibilities of each party. For instance, see the note from the *Commission Nationale de l'Informatique et des Libertés* (CNIL) on the control of Internet use at work [16].

However, the legal sphere only provides a guarantee of judicial sanction for the misuse of technology. It should not be forgotten that a person who deliberately wants to abuse these technologies is able to do so. Let us recall the obvious: this is deeply linked to individual morals that legal sphere cannot entirely control, and the only solution to solve this problem is to encourage personal moral reflection and healthy questioning. Nevertheless, the existence of this misuse can have a serious impact on society. Therefore, as far as possible, some means must be found to prevent the upcoming of abuses, and why not within technology itself.

In this thesis, we are interested in the question of data privacy that arises with this technologies. In particular, in the technical answers that help to protect this privacy (for instance, see the concept of Privacy-by-Design [1]). But before considering any answer, let's try to deepen our knowledge of the issues.

The question of data privacy

It is not a trivial reflection to consider the reasons and limits within which the privacy of data must be guaranteed. Without wishing to contribute to this reflection, we would like to list

some common cases in which data privacy is considered important or even essential.

A first case, and we believe it is essential, is the right of every human being to have its intimacy respected (Article 12 of United Nations Universal Declaration of Human Rights, Article 9 of the French Civil Code ...).

Another case is the protection of sensitive information. Some examples of such are: the precise location of a nuclear arsenal, or critical diplomatic discussions. If the privacy of the data carrying this type of information is not guaranteed, the safety of people is at risk.

The last case we are discussing here concerns the economic field. Namely, the competitive advantages provided by certain technical knowledge and/or expertise that are materialized by an accumulation of data. In most cases, a company's competitive advantages help it to remain profitable, and thus enable it to meet certain needs of its employees.

These common cases are only examples of where data privacy is indeed important. They nonetheless indicate that it would be irresponsible not to look for ways to prevent data misuse. Indeed, although a technical solution cannot be a cure to the underlying problem, it can nevertheless be an aid to the prevention of greater ills. In this thesis, we are interested in a technical answer called *cryptology*.

Cryptology: a technical answer to data privacy

Cryptology is, in its etymological meaning, the science of secrecy. Its subject of study is larger than the simple confidentiality of information. Its purpose is to propose some tools and best practices to guarantee integrity, authenticity and privacy of information to protect.

Historically, the secrecy of information was more closely related to art than science. And its uses were more or less restricted to the military and political spheres. With the advent of digital technologies, the need to protect information changed scale and cryptology democratized itself.

In practice, this science is divided into two disciplines: cryptography and cryptanalysis. Cryptography is about protecting information (etymologically: writing of a secret), and cryptanalysis is about analyzing the secret (etymologically: analysis of the secret). The first seeks to protect information and the second to discover it. This permanent opposition is a source of improvement and refinement of cryptographic tools [2, 3]. In this thesis, we are more interested in the cryptography field, but we hope the reader understands that these two disciplines should not be considered independently.

Securing the exchange of information. The basic principle of cryptography is to scramble information to make it unreadable or uninterpretable to unauthorized persons. This is called an *encryption* process and the resulting information is *encrypted*. This masking of exchanged information is done using a secret element shared by the right holders called a *secret key*. The same secret element is also used to retrieve the original information from the encrypted domain. This is called a *decryption* process. Encryption and decryption are usually called the *primitives* of a cryptographic scheme. This common secret is actually the basis of what we call *symmetric cryptography*. Some examples of symmetric cryptographic schemes are DES, AES, Trivium, etc..

With the advent of telecommunication, the issue of exchanging this common secret arose. Indeed, the secret had itself to transit through telecommunication channels that are insecure. In the 1970s, this problem gave rise to the notion of *asymmetric cryptography*. In that case, the information is scrambled using a public element called a *public key*, and the information is

retrieved using a *private key*. The security of the information is based on some mathematical problems that are hard to solve in the general case but that become easier with a trapdoor. In a nutshell, the public key creates an instance of a mathematical problem by scrambling the information, and the private key is the trapdoor given to a right holder to solve the mathematical problem and hence to retrieve the information. Asymmetric cryptography is less efficient than its symmetric counterpart with respect to communication cost. This is due to an inherent growth in information size while scrambling that does not occur in symmetric cryptography. In practice, asymmetric cryptography is used for secret sharing and authentication protocols but not for the proper exchange of information. Some examples of asymmetric cryptographic schemes are RSA and ElGamal.

Preserving privacy during data processing. Faced with the constant increase in the cost of maintaining IT infrastructures, the market for centralized IT services is growing rapidly. The question of outsourcing the storage and processing of private data is therefore highly relevant. Symmetric and asymmetric cryptography only allow to secure the exchange and storage of information. But what is required in this case is to guarantee the confidentiality of the data also during their processing.

In 1978, Rivest, Adleman and Dertouzos introduced the notion of *homomorphic cryptography* that would allow such data processing. The principle is to have a decryption process that preserves some structure between the encrypted and clear domains allowing the definition of equivalent operations. It is only in the late 2000s that *homomorphic cryptography* has truly emerged with a theoretical solution.

Since then, many research works have improved the practicality of this new type of cryptography. Firstly, by theoretical simplifications of these cryptographic schemes, and secondly, by algorithmic optimizations for efficient implementations.

Nevertheless, this cryptography paradigm seems to have inherently large memory and computational complexities. To give an idea of the order of magnitude, an homomorphic ciphertext is around 10^5 times larger than the data it encrypts, and an operation in the encrypted domain is around 10^6 times heavier than its clear counterpart. As a result, the actors involved in research on the practicality of this new cryptography are in needs of further optimizations such as dedicated hardware acceleration.

The principle of hardware acceleration is to build a dedicated computing architecture for the realization of a specific task. This has to be considered in opposition to a general purpose computing architecture, which is designed to be efficient in the average case of algorithmic needs for a wide variety of applications. This thesis is a contribution for designing such dedicated hardware architectures for homomorphic encryption acceleration.

This thesis

Homomorphic cryptography is a field that is still emerging. Although many applications are being considered, few have the merit of being explored, and even fewer are already feasible on a large scale. Nevertheless, research is making fast-paced progresses towards its use outside the laboratories. And we hope that some of the contributions associated to this thesis may help this development.

Before mentioning our scientific contributions, we will detail a bit the general problematic of homomorphic cryptography and that of its hardware acceleration. From this we will present the results of our work.

The problematic

The main goal of research on homomorphic cryptography is to define an encryption scheme that has encrypted equivalents for all the operations that can be applied on data. This goal is referred to as *Fully Homomorphic Encryption* (FHE). The fact that it took almost 30 years between the introduction of the concept and the first theoretical construction of an FHE scheme is no coincidence. Indeed, two intersecting theoretical problems exist.

The first is the difficulty of finding expressions of mathematical problems that allow homomorphic encryption schemes to be constructed. Just like for asymmetric cryptography, these mathematical problems ensure the security of the schemes. But they also influence the algebraic structures of the cleartext and ciphertext spaces, and hence the construction of equivalent operations.

The second is the necessary presence of an encryption noise. But the noise amplitude in the ciphertexts grows with operations in the encrypted domain. Above a certain noise threshold, the decryption process stop working.

With the research that followed the first theoretical construction of an FHE scheme, some intermediate solutions brought the usages of homomorphic encryption close to being practical at large scale. Nevertheless, the research on hardware acceleration for these schemes has to cope with two main issues.

The first is the variety of scheme propositions. In less than 10 years, four generations of schemes have already been proposed. Hence, their respective maturities and advantages could not be fully and objectively compared over such a short period of time. And of course, each has its own algebraic characteristics which makes difficult, if not impossible, the definition of a computing architecture that would accelerate homomorphic encryption in general. The result is therefore a more qualitative positioning in the choice of schemes to consider for hardware acceleration. During the preparation of this thesis, our analysis of the state-of-the-art of homomorphic cryptography made us consider a particular scheme named after its authors Fan and Vercauteren (FV). It appears as one of the main representatives of the second generation of HE schemes and is well accepted in the FHE community.

The second problematic is that the most mature HE schemes from our point of view have their parameterization dependent on the applications to be performed in the encrypted domain. And of course, it would have been too easy if the parameter ranges were small. This is due to the schemes' noise growth which requires the noise gauge to be sized properly. In the case of the FV scheme, the handled elements are polynomials with large degree and large coefficients. Both degree and coefficients get significantly larger with the noise absorption capacity.

Scientific contributions

During this work, we have particularly addressed the issue of the wide range of the sizing parameters. The literature has proposed the basic elements upon which are built our contributions. Namely, the use of Residue Number System (RNS) to tackle the complexity brought by large integer arithmetics, and the use of Number Theoretical Transforms (NTT) to reduce the asymptotic complexity of polynomial multiplications. Building on this, we argue that the coupled approach of RNS and NTT allows the definition of hybrid computing systems, which promise a significant acceleration of encrypted-domain computing with the FV scheme.

This thesis results from the in-depth analysis of FV as well as contributions to solve the main difficulties of combining the use of RNS and NTT for large parameters.

Our main contributions are therefore the following. First, an analysis of the RNS/NTT-coupled approach and a theoretical validation of its feasibility up to very large parameter sets. Second, the exploration of GPU acceleration of some RNS specific algorithms brought by concomitant works for the full adaptation of FV to RNS. Third, the design and a proof-of-concept implementation of a data flow oriented hardware for efficient residue polynomial multiplications, and the study of its capability to scale.

This last contribution is the subject of particular attention in order to overcome certain design problems that have previously limited the feasibility of the coupled approach at large scale. In particular, we propose a solution to locally pre-compute the values necessary for NTT computations without impacting the operations' throughput. This is achieved with two contributions: the design of a fully-streaming multi-field NTT, and the design of a twiddle factor set generator. Finally, to take into account the issue of significant changes in FV parameters w.r.t. an application, the automatic generation of RTL descriptions for the proposed designs has been explored.

Outline of this document

This document begins by positioning our work within the state-of-the-art of homomorphic encryption acceleration. First, we identify a subset of well-accepted schemes at the time of this thesis begun, and present their implementation issues. It is followed by a discussion on the existing solutions and related works addressing these issues. The first chapter ends with our arguments in favor of our decision to make the FV scheme the subject of an in-depth study to propose a hardware acceleration strategy consistent with the wide ranges of its parameters.

In the second chapter, the FV encryption scheme is detailed. The study of its performance profile orients us towards an implementation strategy based on the RNS representation and NTT-based polynomial multiplications. This strategy is theoretically validated for the wide spectrum of FV parameters. Analyzing closely the related work on a fully RNS version of FV allowed us to refine this strategy by addressing both RNS specific functions and NTT-based Residue Polynomial Multiplication (RPM).

The third and fourth chapters provide details on basic hardware blocks for the efficient design of NTT-based RPM. Both concern the problem of managing pre-computed values for NTT. The third chapter deals more specifically with the automatic generation of RTL circuits for data-flow multi-field NTTs with a particular focus on the handling of the twiddle factor sets. The fourth chapter presents an on-the-fly generation approach of these twiddle factors satisfying the throughput requirements imposed by the NTT circuits.

Based on the analysis and contributions of previous chapters, the fifth chapter presents our proposal of an hybrid computing system for the acceleration of FV. Firstly, the GPU acceleration of RNS-specific functions is studied. Secondly, an architecture's proposal for RPMs is detailed. Together these two studies allow a performance projection for an hybrid GPU/FPGA computing system.

Finally, a general conclusion summarizes the present work and offers perspectives.

Chapter 1

Review of homomorphic encryption and its practicability

This chapter presents a review of homomorphic cryptography and its implementations. The primary objective is to position ourselves within the research activity around Homomorphic Encryption (HE). As far as possible, we present the elements that make our qualitative choices understandable. The second objective is to introduce the notions that appear in the various discussions in this document as arguments for our choices.

After having introduced the basic notions of homomorphic encryption, a state of the art of the HE schemes is presented, and our positioning is expressed. The detailed presentation of the LWE and RLWE problems makes it possible to present the research for using HE in practice. Finally, we highlight the implementation issues that must be addressed and we summarize the related works on this matter.

1.1 Introduction

1.1.1 Technical introduction to homomorphic encryption

HE schemes are able to perform operations on encrypted data without decrypting them first. This ability comes from choosing a homomorphism as the decryption function of the encryption scheme. The definition of a homomorphism directly highlights the desired feature: it is a *structure-preserving map* between two algebraic structures of the same type (groups, rings, fields ...). Considering the plaintext space and the ciphertext space as algebraic structures, having a homomorphism is then to have a map that expresses an equivalence between some operations in the encrypted and clear domains.

To summarize, a HE scheme has its decryption function to be a homomorphism. Given an algebraic operation $\circ$ over the ciphertext space and an algebraic operation $*$ in the plaintext space, the decryption function is a homomorphism for these operations if and only if:

$$\mathrm{Dec}(\mathtt{ct}_1 \circ \mathtt{ct}_2) = \mathrm{Dec}(\mathtt{ct}_1) * \mathrm{Dec}(\mathtt{ct}_2), \text{ for all } \mathtt{ct}_1, \mathtt{ct}_2 \text{ in ciphertext space.}$$

It is said that an equivalent of $*$ exists in the ciphertext space by decryption homomorphism.

1.1.2 Basic notions for homomorphic encryption

This subsection presents some basic notions for the proper understanding of the homomorphic cryptography's state-of-the-art.

Semantic security. This notion has been introduced in 1982 by Goldwasser and Micali [17] and further detailed in 1984 [18]. It states that it should be unfeasible to efficiently retrieve information about messages from the knowledge of a polynomial number of ciphertexts and the public elements of the encryption scheme. This is analogous to information theoretic security w.r.t. a computationally-bounded adversary.

More precisely, the semantic security notion is equivalent to *ciphertext indistinguishability* under chosen-plaintext attack. Meaning that an attacker is not able to distinguish two ciphertexts encrypting the same message. In particular, Goldwasser and Micali show that *probabilistic encryption* may be needed for that purpose, which is the case for homomorphic cryptography. It is well known that HE schemes are necessarily probabilistic, and this has two main consequences.

First consequence: there is a significant data size expansion between clear and encrypted data. Because the ciphertext space has to be significantly larger than the plaintext space to insure ciphertext indistinguishability. Thus, with the help of a well-constructed probabilistic encryption, the upcoming of exploitable patterns in ciphertexts should be highly unlikely, even after publicly known operations over encrypted data.

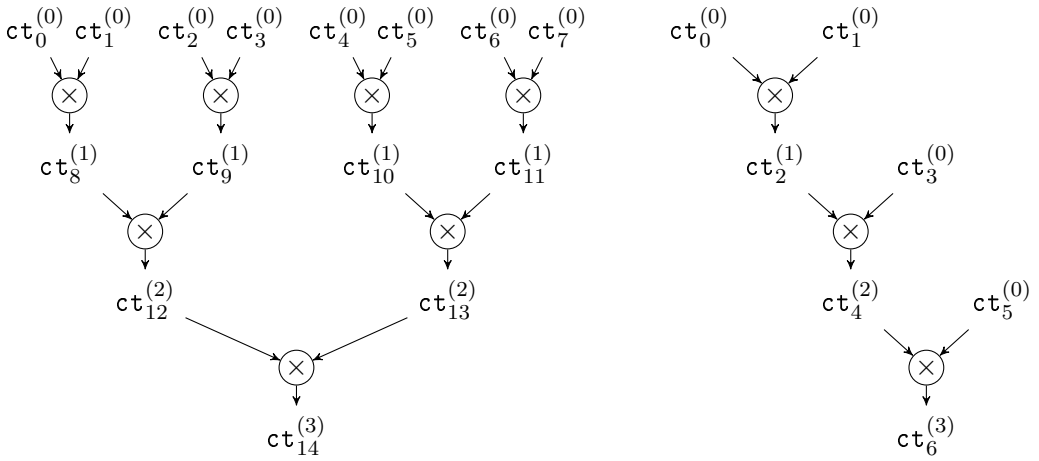
Second consequence: the ciphertexts are noisy because probabilistic encryption adds a noise during the encryption. As already mentioned, this is one of the main issue for the construction of practical homomorphic schemes. More details follow along with the definition of the correctness notion.

Correctness. A homomorphic scheme is correct if the decryption function always gives the appropriate plaintext. In the presence of noisy ciphertexts, the decryption function acts as a filter that erase the noise under a certain noise threshold. The problem with homomorphic encryption is that operations in the encrypted domain make the noise term to grow. However, the decryption function is able to erase only a given amount of noise, and becomes random otherwise. Until recent works (onwards 2014), no homomorphic scheme has been found with an efficient enough bootstrapping procedure to perform encrypted operations that do not increase the noise.

The noise growth during a ciphertext operation is dependent on operands' noise level. For a generic study of the noise constraints for parameter selection, it is suitable to consider a worst-case and an optimal scenarios. The worst-case scenario considers a binary tree of operations from fresh ciphertexts towards a depth L operation (Figure 1.1a). Hence, the noise level of the operands is at each level the maximum possible. The optimal scenario considers that every operation is performed with one of the operand being a freshly encrypted ciphertext (Figure 1.1b).

Homomorphic cryptography terminology. The main goal of homomorphic cryptography is the definition of Fully Homomorphic Encryption (FHE) schemes. A FHE scheme has an encrypted equivalent for all applicable functions in the clear domain. However, the coupled difficulty of defining schemes with appropriate homomorphism properties while handling noisy ciphertext makes things a little tricky. Intermediate solutions are easier to build and, as a result, different terminologies appear.

Partially Homomorphic Encryption (PHE) qualifies schemes with homomorphism properties only for functions composed of a subset of the plaintext space's algebraic operations. For example, if the decryption function is a homomorphism of additive group (respectively multiplicative group), hence only functions composed of additions (respectively multiplications)



(a) Binary tree of multiplications

(b) Optimized circuit for multiplication

Figure 1.1: Tree models that consider the worst and optimal scenarios regarding the noise growth for a degree L function in the encrypted domain. The considered operation is the ciphertext multiplication and the multiplicative depth L is equal to three. The exponents in brackets are the ciphertext depths with respect to multiplication.

have an equivalent in the encrypted domain.

Somewhat Homomorphic Encryption (SHE) is the term that qualify schemes that have their decryption function to be at least a ring homomorphism. It means that both plaintext additions and multiplications have equivalents in the encrypted domain. Nevertheless, they are not FHE schemes due to the noise threshold limiting their evaluation capability.

Now that the basic notions have been settled, the next section presents the state-of-the art of homomorphic cryptography.

1.2 State of the art of homomorphic cryptography

1.2.1 History towards FHE

The concept of Homomorphic Encryption (HE) has been introduced by Rivest, Adleman and Dertouzos in 1978 under the term of *privacy homomorphisms* [19]. This work has followed the highlight of multiplicative homomorphism property of the first version of the RSA cryptosystem. Its main motivation was to bring together what seems to be two irreconcilable services: sensitive data exploitation and time-sharing data bank. This initial paper has settled fundamental and desirable features of privacy homomorphisms. It has also given some basic yet insecure constructions of such desirable schemes.

In particular, Rivest et al. has noticed that a scheme that is homomorphic for predicates (like comparisons) may become insecure under some simple assumptions on the data bank side. This was mostly due to *deterministic encryption* that makes the retrieval of information about messages easier when exploiting known properties of the encryption scheme (set ordering, test to zero, etc.). The notion of *semantic security* from Goldwasser and Micali has brought a theoretical solution to this problem. Nevertheless, both important data size expansion and noisy ciphertext has come along with probabilistic encryption.

At this point, an open problem for homomorphic cryptography was to define secure encryption scheme that is able to evaluate in the encrypted domain all functions applicable in the plaintext domain. This desired goal is called Fully Homomorphic Encryption (FHE). Both problematics of finding an appropriate decryption homomorphism and handling noisy ciphertext have been inherently linked since.

Between 1978 and 2009, the research have proposed only imperfect solutions regarding that goal, although some of them are sufficient for some kind of applications. In particular, the works essentially proposed Partial Homomorphic Encryption schemes. Additive homomorphic schemes have been constructed like the Goldwasser-Micali encryption scheme [17] in 1982 or the Pallier encryption scheme [20] in 1999. As well as multiplicative homomorphic schemes. For instance, the Elgamal encryption scheme [21] in 1985, based on the Diffie-Hellman key exchange.

Other works have dealt with the definition of Somewhat Homomorphic Encryption schemes. The trouble was to make both addition and multiplication possible in the encrypted domain. A noticeable work in that area is the one of Boneh, Goh and Nissim in 2005 [22]. They made possible an infinite number of additions plus one multiplication in the encrypted domain.

The first SHE scheme is brought by Aguilar-Melchor, Gaborit and Herranz in 2008 [23], and improved in 2010 [24]. They allow the evaluation of a circuit of multiplicative depth d at the cost of a ciphertext size's growth exponential with d.

Finally, the very first theoretical Fully Homomorphic Encryption scheme is brought by Gentry in 2009. It is based on a SHE scheme using a noise management technique called *bootstrapping*. This technique theoretically avoid an infinite expansion of the ciphertext size for homomorphic evaluation by reducing the level of noise. To achieve this, Gentry proposes to homomorphically re-encrypt a ciphertext to homomorphically decrypt it just after. This reduces the noise as the decryption function is a noise filter, and this is theoretically secure because everything happen in the encrypted domain.

Even if his initial construction was impractical, Gentry's work is commonly viewed as the FHE breakthrough. In this thesis, we have only considered schemes that resulted from the works of Aguilar-Melchor et al. and of Gentry. At the time of writing, it is classic to consider four generations of Homomorphic Encryption schemes. The next subsection presents these four generations.

1.2.2 Four generations of FHE schemes

First generation: bootstrapping. Before Gentry's paper of 2009 [25], PHE and SHE schemes were known, but no concrete solutions to build FHE schemes existed. Gentry has based his construction of the first FHE scheme on what he called a *bootstrapping* procedure: it consists in evaluating the scheme decryption function in the encrypted domain. A SHE scheme having sufficient noise gauge for bootstrapping plus at least another encrypted operation (addition or multiplication) becomes a FHE scheme.

Gentry proposes to see the decryption function as a noise reducing procedure. By over-encrypting a ciphertext and homomorphically apply the decryption function, the result is still encrypted but with a "fresh" noise level. Gentry assumes that the scheme is still secure with a public encrypted version of the private key. This security assumption is called *circular security*. Further SHE/FHE schemes still rely on similar assumptions. All known noise management techniques require somehow a partial and/or a masked knowledge of the secret.

The first implementations of bootstrapping-based FHE schemes were impractical [26, 27]. This is due to the important noise growth of the chosen SHE and their complex decryption circuit. The size of the scheme elements and the primitives' performance are truly not as competitive as those of further generations. For example, the smallest parameter set for the best implementation of Gentry's scheme [27] results in a public key of 17 Mbyte and bootstrapping procedure in 6 second on an high-end server system. Hence, we mention these schemes for completeness but our discussions will not consider them anymore.

Nevertheless, Gentry's initial work has opened the gate to numerous advances in the definition of more efficient SHE and FHE schemes.

Second generation: Leveled-FHE (L-FHE). The schemes of the second generation are not *stricto sensus* fully homomorphic, but can nevertheless address applications with an arbitrary prior-fixed complexity. This is due to less radical noise management techniques than bootstrapping that keep the noise under the threshold limit a fixed number of times. The parameters of these schemes are chosen for a given evaluation capability in the encrypted domain, characterized in practice by the multiplicative depth of the considered application. This is because the noise growth during additions is negligible compared to its growth during multiplications. The common qualification for this kind of schemes is Leveled-FHE.

The initial techniques are called *modulus switching* or *key switching* and appear with the works of Brakerski and Vaikuntanathan [28, 29]. In particular, the *modulus switching* technique became the angular stone of a well accepeted scheme called BGV [30]. A ladder of moduli is used to control the noise level in the ciphertexts. A ciphertext is constructed with element modulo the product of all the modulus in the ladder. After a multiplication, a modulus is simply "removed" of the ladder by scaling down the ciphertext. Doing so reduces proportionally the ciphertext's noise making room to perform further multiplications. With this technique, the noise growth is linear with the multiplicative depth of the evaluated function; in contrary to a quadratic noise growth in concomitant SHE schemes.

A disadvantage of having a moduli ladder is that the size of a ciphertext varies along the operations in the encrypted domain. In [31] Brakerski observed that the desired linear noise growth could be achieved without a moduli ladder. The resulting schemes are called *scale-invariant*. Brakerski constructs the first scale-invariant scheme upon a problem called the Learning With Errors problem (LWE). A scheme builds upon a ring variant of the LWE problem (RLWE) is quickly proposed by Fan and Vercauteren (FV [7]). The FV scheme is still now a well-accepted candidate for practical HE applications.

Third generation: conceptual simplifications for L-FHE. The third generation revisits the second generation to simplify the construction of Leveled-FHE schemes. They avoid the need of extra primitives which purpose are to limit the noise growth or to restructure ciphertexts due to non-canonical scheme primitives. Among the works from this generation, the initial GSW [32] and SHIELD [8] are the most popular. They are both based on Brakerski schemes [31].

From a practical point of view, third generation schemes are more specialized than those of second generation. They have a highly reduced noise growth considering applications that only require multiplications with one of the operands being a fresh ciphertext. This scenario has been presented in Figure 1.1b when presenting the correctness notion. Hence, the schemes of the third generation are promising candidate for the applications that fit in this scenario.

Fourth generation: gate bootstrapping. The fourth generation revisits the bootstrapping approach [33, 34, 35, 6]. This generation makes bootstrapping being inherently linked with an arithmetic operation rather than solely a noise-reducing procedure. In a nutshell, any arithmetic operation results in a fresh ciphertext. Hence, the level of noise is not cumulative anymore because the ciphertext is always a fresh one. Some works call this approach *gate bootstrapping*, in reference to the equivalent bitwise operations considering a binary message space.

This generation is really promising for a generalized use of homomorphic cryptography, and in particular the TFHE [6] scheme. This scheme evaluates its bootstrapped gate in the order of tenth of milliseconds. However, its relative youth makes difficult the qualification of the proper maturity and requires the exploration of efficient implementation strategies before considering dedicated hardware implementations.

1.2.3 Additional considerations and positioning

At first, the research's effervescence around homomorphic cryptography can be confusing for whom is interested in practical considerations on SHE/FHE. Nevertheless, when looking for maturity, it seems that LWE-based schemes (and in particular those over its ring variant RLWE) are currently the most balanced in terms of security and efficiency.

Indeed, the improvements of the second generation, and later of the third and fourth generations, were mainly dependent on the upcoming of the Learning With Errors (LWE) problem. This problem is introduced by Regev in 2005 [36] and formalized by the same author in 2010 [37]. The LWE-based schemes often handle matrices and vectors of integer elements. Its ring variant RLWE makes the schemes to handle matrices and vectors of polynomial ring elements.

The fourth generation also constructs its schemes over the LWE problems. But its most promising scheme TFHE [6] has chosen an algebraic structure slightly different from others to improve primitive performances. In particular, it considers polynomials with real coefficients modulo one. Consequently, it does not face up the same implementation problems, and hence requires different implementation approaches.

The literature considers also other mathematical problems for the HE context. Since the second generation, some works [38, 39, 40] have constructed schemes upon NTRU [41] and NTRU' [42] hardness. These problems are related to the search for the shortest vector in a lattice under some additional security assumptions. Due to attacks exploiting subfields' presence in some overstretched versions of these NTRU assumptions [43, 44], NTRU-based schemes are not considered sufficiently secure for the homomorphic cryptography context. Indeed, in order to be competitive, these schemes require choosing parameters that put them dangerously closer to simpler resolution of the underlying NTRU problem.

Another family of cryptosystem regroups a list of improvements over a scheme introduced by van Djik et al. [45] in 2010. Unlike the other schemes, which are more or less related to lattices, they rely on a problem over the integers called the *Approximate Greatest Common Divisor* (AGCD) problem. These schemes [46, 47, 48, 49] are conceptually simpler than those over lattices. However, they do not have the same practical efficiency, in particular regarding the size of the handled keys and ciphertexts. For example, in the scale-invariant scheme proposed by Coron et al. [48], for a reasonable security setting ($\lambda = 80$) the public key size is roughly 100 GByte and the key generation took 213 hours on a high-end server system.

The current literature makes us consider that RLWE-based schemes are the closest to concrete utilization. In particular, the FV scheme [7] (2^{nd} generation) and the SHIELD [8] (3^{rd} generation) are well accepted as Leveled-FHE schemes. For its part, the leading scheme of the fourth generation (TFHE) seems not to require similar implementation approaches than FV and SHIELD. Nevertheless, we assume unlikely that TFHE totally eclipses FV and SHIELD, because the utilization context for these schemes are not likely to be the same. We are therefore interested in the issues of implementing the HE schemes based on the RLWE problem.

After these considerations on the state-of-the-art of homomorphic cryptography, we now discuss implementation issues. Beforehand, the LWE and RLWE problems are described in order to present their parameters and their algebraic structures.

1.3 The Learning With Errors (LWE) problem and its ring variant (RLWE)

The Learning With Error (LWE) problem and its ring variant (RLWE) have led to major improvements in homomorphic cryptography. They give the most accepted schemes of second and third generations. They allow simpler construction with algebraic structures and operations giving relatively good execution performances. Furthermore, they also give good reasons to believe in their hardness.

1.3.1 The LWE problem

The Learning With Errors problem is introduced by Regev in 2005 [36] and formalized by the same author in 2010 [37]. Regev proves this problem to be secure under quantum-reduction to approximate *general* lattice problems. The reduction proof has been enriched with classical approaches [50, 51]. It means that for some adequate parameters this problem is at least as difficult as some problems over lattices, whose difficulties are quite well accepted. Even if some blurs remain in the concrete choice of parameters, the LWE problem is a promising candidate for post-quantum cryptography. Most of the underlying descriptions follow the survey of Regev in [37] and thesis works of Migliore [52] and Bonnoron [9].

Introducing example. A particular formalism describes the LWE problem as the task of recovering a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ given a sequence of *random approximate linear equations* on $\mathbf{s}$. To clarify this description, here is the example given in Regev's survey.

Consider vectors of dimension $n = 4$ with elements uniformly sampled modulo $q = 17$, and consider also an error distribution χ on ± 1, For a given secret $\mathbf{s} \in \mathbb{Z}_{17}^4$, a new random approximate linear equation is produced as follow. Sample uniformly an element $\mathbf{a}$ over $\mathbb{Z}_{17}^4$, sample an error e from χ, and express equation over the variable $\mathbf{x} \in \mathbb{Z}_{17}^4$:

$$\langle \mathbf{a}, \mathbf{x} \rangle \bmod 17 \approx (\langle \mathbf{a}, \mathbf{s} \rangle + e) \bmod 17.$$

The problem instance consists in finding $\mathbf{x} = \mathbf{s} = (s_1, s_2, s_3, s_4) \in \mathbb{Z}_{17}^4$ given an arbitrary number w of random approximate linear equations. For instance with $\mathbf{s} = (0, 13, 9, 11)$.

$$
\begin{aligned}
14s_1 + 15s_2 + 5s_3 + 2s_4 \pmod{17} &\approx 8 \\
13s_1 + 14s_2 + 14s_3 + 6s_4 \pmod{17} &\approx 16 \\
6s_1 + 10s_2 + 13s_3 + 1s_4 \pmod{17} &\approx 3 \\
3s_1 + 6s_2 + 4s_3 + 5s_4 \pmod{17} &\approx 16 \\
&\vdots \\
6s_1 + 7s_2 + 16s_3 + 2s_4 \pmod{17} &\approx 3
\end{aligned}
$$

The hardness of this problem is function of n, q and the probability distribution χ that sample the error. The following paragraphs detail these notions with a more formal definition of the LWE problem.

LWE definition. Let $n \geq 1$ and $q \geq 2$ be two integers, and χ a probability distribution over the set of integers $(-q/2, q/2]$ (noted by convention $\mathbb{Z}_q$). The naming convention refers to n as the dimension, q as the modulus and χ as the error distribution of the LWE instance. A classical choice for χ is a zero centered normal distribution of standard deviation σ rounded to the nearest integer. An LWE instance is then defined by its three parameters n, q, and σ (further called error size).

Given $\mathbf{s} \in \mathbb{Z}_q^n$, let $A_{\mathbf{s},\chi}$ be a probability distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$. This distribution is obtained by following the process hereafter: choose a vector $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choose an integer $e \in \mathbb{Z}_q$ according to χ, and then output $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Note that elements given by $A_{\mathbf{s},\chi}$ are approximate linear equations similar to those in the simple example of previous paragraph.

Considering the distribution $A_{\mathbf{s},\chi}$, it is said that an algorithm solves (n, q, σ)-LWE if, for any $\mathbf{s} \in \mathbb{Z}_q^n$ and given an arbitrary number of independent samples from $A_{\mathbf{s},\chi}$, it outputs $\mathbf{s}$ with high probability. Among all known algorithms solving the LWE problem, none require less than $2^{O(n)}$ memory or time complexity for correctly chosen (n, q, σ). Meaning that it is possible to construct LWE instances that cannot be solved given a computationally bounded adversary. At least in the present state of our knowledge concerning the difficult underlying problems. Among the reasons to believe in the hardness of the LWE problem, the most important are the proven reductions of LWE hardness to worst-case hardness of standard lattice problems [36, 50, 51]. The hardness of the latter's problems is well accepted considering their resistance to research efforts to solve them.

Simple scheme construction. Upon this problem, Regev proposed a simple asymmetric scheme that is detailed here. The purpose is to understand how the LWE problem could be used to construct an encryption scheme. The scheme is called REG in this description.

Let (n, q, σ) be an LWE instance hard to solve for a given number w of approximate linear equations.

- REG.SecretKeyGen(λ): sample $\mathbf{s}$ from $\mathbb{Z}_q^n$, and output $\mathtt{sk} = \mathbf{s}$.
- REG.PublicKeyGen($\mathtt{sk}$, w): sample w vectors $\mathbf{a}_1, ..., \mathbf{a}_w$ uniformly from $\mathbb{Z}_q^n$ and w error offsets $e_1, ..., e_w$ from χ.
  Return $\mathtt{pk} = \{(\mathbf{a}_i, b_i)\}_{i=1}^w$, with $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle / q + e_i$.
- REG.Encrypt($\mathtt{pk}$, w, $m \in \{0, 1\}$): to encrypt a binary message m, choose a random subset S of the w public key elements $\{(\mathbf{a}_i, b_i)\}_{i=1}^w$.
  Compute $\mathbf{c}_0 = \sum_S \mathbf{a}_i$ and $\mathbf{c}_1 = m/2 + \sum_S b_i$.
  Return $\mathtt{ct} = (\mathbf{c}_0, c_1) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

- REG.Decrypt(sk, ct): compute $r = \mathbf{c}_1 - \langle \mathbf{c}_0, \mathbf{s} \rangle / q$.
  Return $m = 0$ if r is closer to 0 than to $1/2$, return $m = 1$ otherwise.

The proof of correctness follows from an appropriate choice of parameters and some probability analysis (see [36]).

Choice of parameters. As introduced in previous paragraphs, the hardness of the LWE problem (and hence the security of the schemes) is dependent of its parameters (n, q, σ). Since 2005, numerous works have refined the practical choice of parameters. This derivation is mainly empirical and relies on some estimation models of all known attacks against LWE. The current state-of-the-art estimator is maintained by Albrecht [53].

In the case of LWE instances for homomorphic encryption, the additional dimension of correctness must be taken into account in the derivation of parameters. The resulting derivation rules have multiple elements to consider making them quite complex at first sight. Here are some simple thumb rules regarding the derivation of parameters.

Security is mainly improved when increasing the dimension n of the underlying problems, but n has also an error magnifier effect. Indeed, the error size σ should not be too small regarding the dimension ($\sigma > \omega(\sqrt{n})$) in order to effectively mask enough information for the LWE problem to be difficult. Correctness is improved when the noise is very small in front of the modulus ($\sigma \ll q$). Nevertheless, larger q decreases security for a given n [37] ($q < 2^{poly(n)}$). Thus when going for better correctness, one has to be careful to not reach the upper bound on q or the lower bound on σ.

We note that this refinement of parameters remains an active research area, and in particular for more specialized version of LWE like the Learning With Errors over Rings (RLWE).

From a practical point of view, the LWE-based schemes have rather large elements to handle. Thus, their practical uses tend to be expensive in terms of memory usage, performance and communication cost. These practical issues are particularly noteworthy for FHE applications as the parameters must be large enough for correctness. These are the reasons for the FHE community to explore LWE problem specializations to more structured algebraic structures like polynomial rings.

1.3.2 The LWE problem over rings

The Learning With Errors over Rings (RLWE) is a specialized version of the Learning With Errors problem to polynomial rings over finite fields. This specialization allows the addition of extra-algebraic structures in order to improve the efficiency of cryptographic schemes built upon the problem. The RLWE problem is formalized by Lyubashevsky et al. [54] in 2010.

RLWE definition. The hereafter definition of RLWE is slightly different from [54], and this should be taken into account for rigorous hardness analysis [55]. Nevertheless, this has only an influence on the parameter selection (in particular in the noise distribution). The two definitions are equivalent from an computational perspective.

Let R be a ring of degree n over the integers, q a positive integer and χ a probability distribution over $R_q = R/qR$, the quotient ring defined by the modulus q. For simplicity, consider R to be the set of polynomials with integer coefficients provided with polynomial addition and polynomial multiplication. Similarly, consider R_q as R with integer coefficients modulo

q. The distribution χ may be seen as sampling n coefficients from a normal distribution over $[-q/2, q/2[$, with standard deviation σ, to construct a polynomial of R_q.

Given $\mathbf{s} \in R_q$, let $A_{\mathbf{s},\chi}$ be the probability distribution over $R_q \times R_q$ obtained by following the process hereafter. Choose an element $\mathbf{a} \in R_q$ uniformly at random, choose an element $\mathbf{e} \in R_q$ according to χ, and then output $(\mathbf{a}, (\mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \bmod q) \in R_q^2$.

Considering the distribution $A_{\mathbf{s},\chi}$, it is said that an algorithm solves (n, q, χ)-RLWE if, for any $\mathbf{s} \in R_q$ and given an arbitrary number of independent samples from $A_{\mathbf{s},\chi}$ it outputs $\mathbf{s}$ with high probability. The decision version of the problem consists in distinguishing samples from $A_{\mathbf{s},\chi}$ from uniform samples in R_q^2.

Simple encryption scheme. When formalizing the problem, Lyubashevsky et al. [54] present a simple asymmetric encryption scheme called here LPR.

Select a polynomial ring $R = \mathbb{Z}[X]/(F(X))$, a modulus q and a probability distribution χ, with ($n = \deg(F)$, q, σ) depending on the security parameter λ. Choose also a plaintext modulus t, defining the ring R_t in which the message will be encoded before encryption.

- LPR.SecretKeyGen(λ): sample $\mathbf{s}$ from χ, and output $\mathtt{sk} = \mathbf{s} \in R_q$.
- LPR.PublicKeyGen($\mathtt{sk}$): sample $\mathbf{a}$ uniformly from R_q, $\mathbf{e}$ from χ.
  Compute $\mathbf{p}_0 = [-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})]_q$ and set $\mathbf{p}_1 = \mathbf{a}$.
  Return $\mathtt{pk} = (\mathbf{p}_0, \mathbf{p}_1) \in R_q^2$.
- LPR.Encrypt($\mathtt{pk}$, $\mathbf{m} \in R_t$): Let $\Delta = \lfloor \frac{q}{t} \rfloor$, and $\mathbf{u}$, $\mathbf{e}_1$, $\mathbf{e}_2$ being samples from χ.
  Compute $\mathbf{c}_0 = [\mathbf{p}_0\mathbf{u} + \mathbf{e}_1 + \Delta\mathbf{m}]_q$ and $\mathbf{c}_1 = \mathbf{p}_1\mathbf{u} + \mathbf{e}_2$.
  Return $\mathtt{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in R_q^2$.
- LPR.Decrypt($\mathtt{sk}$, $\mathtt{ct}$): Return $\mathbf{m} = \left\lceil \lfloor \frac{t}{q} \cdot [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \rceil \right\rfloor_t$.

To express the correctness requirement, let consider that $[\mathbf{c}_0 + \mathbf{c}_1\mathbf{s}]_q = [\Delta[\mathbf{m}]_t + \mathbf{v}]_q$. The polynomial $\mathbf{v}$ is commonly called the noise term of the ciphertext $(\mathbf{c}_0, \mathbf{c}_1)$. It follows that the decryption is correct if $\|\mathbf{v}\|_\infty < (\Delta - r_t(q))/2$, with $r_t(q) = t(q/t - \Delta)$.

RLWE hardness. In the initial paper of Lyubashevsky et al. [54] formalizing the RLWE problem, the authors prove the hardness of search-RLWE by a quantum reduction of worst-case problems over *ideal* lattices. In a recent paper Peikert et al. [56] give an equivalent result for decision-RLWE over which most of SHE schemes rely on. Nevertheless, the RLWE problem and its parameters characterization are not as well explored as for LWE.

Parameterization for RLWE-based schemes often extrapolates that of LWE-based ones. Thus, it does not take into account the specificities of RLWE algebraic structures. Hence, the upcoming of RLWE cryptanalysis changing this parametrization is a potential event to consider. Nevertheless, it seems reasonable that further cryptanalysis will not deeply change the ranges of parameters.

The following paragraph details the algebraic structures of RLWE-based schemes considered for efficient construction of FHE.

Algebraic structures. The RLWE problem defined in [54] is focused on the polynomial rings $R = \mathbb{Z}[X]/(\Phi_m(X))$ of integer polynomials modulo a *cyclotomic polynomial* of order m. By definition, $\Phi_m(X) = \prod_\eta (X - \eta)$ where η ranges over the m-th primitive roots of unity.

The quantum hardness reduction from *ideal* lattice problems has been proven for general cyclotomic field [54], and recently even for any number field [56]. Most RLWE instances choose the $m = 2^k$-th cyclotomic polynomials in practice. It implies that $\Phi_m(X) = X^n + 1$ with $n = m/2$. The attractive complexity brought by a special case of FFT-based polynomial multiplication motivates this choice. These approaches are presented later in this chapter.

In addition to the n resulting from the choice of R, the RLWE problem requires the selection of an integer modulus $q > 1$ that bounds the size of the polynomial coefficients. Accordingly to q, $\mathbb{Z}_q$ defines the set of integer $[-q/2, q/2[$, and this notation is extended to define $R_q = \mathbb{Z}_q[X]/(\Phi_m(X))$ the subset of R with coefficients in $\mathbb{Z}_q$. In practice, the handled elements are viewed as integer polynomials reduced modulo $(\Phi_m(X), q)$. The hardness result of Peikert et al. [56] yield for any modulus q as long as it has the appropriate size regarding n and χ.

The previous description of the LWE and RLWE problems provide a better understanding of the state-of-the-art for the implementation of homomorphic encryption. The objective is to understand our positioning in relation to the multiplicity of works addressing issues at different levels.

1.4 Homomorphic encryption in practice

One of the main implementation issue for homomorphic encryption is the data size expansion resulting from the necessity of semantic security. Another issue is the wide variety of applications in conjunction with only intermediate solutions for encrypted-computing. The second issue is particularly present when considering L-FHE schemes as we do. Without considering any particular application, it is nonetheless important to consider how the schemes and their parameterizations influence the efficiency of an encrypted application.

In this section the literature's solutions to these problems are presented. First, it introduces known techniques to mitigate the impact of the data-size expansion. Then, it presents a typical example of HE scheme comparison and finally exposes the question of choosing a plaintext space w.r.t. a message encoding method.

1.4.1 Mitigating data expansion impact

Data size expansion has two major impacts. First, an important computational overhead compared to a non-encrypted application (roughly millions of times slower). Second, communications are quickly problematic when going for applications with lots of data.

Beside the search for homomorphic schemes with reduced data size expansion, two methods to address this issue are commonly presented. One results from the exploitation of the algebraic structures of some schemes to pack multiple messages in independent ciphertext's "slots". This technique is known as *batching* and addresses the two dimensions of the data expansion problem. The other is known as *transciphering* and "only" addresses the communication overhead. It is based on the ability of an homomorphic scheme to evaluate the decryption function of classical symmetric schemes.

Batching. The batching technique for homomorphic cryptography is introduced by Smart and Vercauteren in 2010 [26] and formalized a little after in [57].

In a nutshell, this technique is the exploitation of the Chinese Remainder Theorem (CRT) to decompose the scheme message space into multiple sub-spaces. Not all the schemes are able to use this technique due to the algebraic structure of their message space. The presentation here mainly refers to schemes based upon the RLWE problem.

Smart and Vecauteren propose to select the polynomial $F(X)$ for the ring R as a monic irreducible polynomial over $\mathbb{Z}_q[X]$ but as a reducible polynomial over $\mathbb{Z}_t[X]$. They present the odd m-th cyclotomic polynomials as possible candidates for $F(X)$. This choice is motivated by their desirable factorization property over $\mathbb{Z}_t[X]$, and in particular for $t = 2$.

When $t = 2$, $\Phi_m(X)$ factors over $\mathbb{Z}_2[X]$ into $r = (\varphi(m))^{\text{I}}/d$ distinct polynomials of degree d, with d being the smallest integer such that $2^d = 1 \mod m$.

$$\Phi_m(X) = \prod_{g=1}^{r} F_g(X) \ , \ F_g \in \mathbb{Z}_2[X] \text{ and } \deg(F_g) = d \text{ for all } g \text{ in } \{1, \ldots, r\}. \tag{1.1}$$

Due to this decomposition, the polynomial ring $\mathbb{Z}_2[X]/(\Phi_m(X))$ is isomorphic[II] to the product ring $\mathbb{Z}_2[X]/(F_1(X)) \times ... \times \mathbb{Z}_2[X]/(F_r(X))$. Moreover, each finite field $\mathbb{Z}_2[X]/(F_g(X))$ is isomorphic to the Galois field of order 2^d. Hence, an element of $\mathbb{Z}_2[X]/(\Phi_m(X))$ may embed up to r elements of any Galois field of order 2^k with $k \mid d$. In particular for $k = 1$, it implies that an element of the message space may embed up to r binary elements in independent "slots". For a more detailed presentation of the technique, please refer to [57].

As this technique only exploits the algebraic structure of the message space without modifying anything in the ciphertext space, it consequently gives a r times speed up at roughly no cost. Nevertheless, it has to be considered that only slot-wise operations are straightforwardly available in the ciphertext space. Some techniques to handle the slots while in the ciphertext domain exist though [58].

The main limitation of this technique is that it is not compatible with an optimization for polynomial multiplications known as Negative Wrapped Convolution (NWC). Indeed, $F(X)$ has to be a power-of-two cyclotomic polynomial for NWC, which is not suitable with batching for binary message. One may find a plaintext space parameter t that allows batching with power of two cyclotomic polynomials. Nevertheless, non-binary message may be inconvenient at another level. This is more detailed in subsection 1.4.2.

To conclude, this technique is a promising approach to reduce the computational and communication overheads. A limitation is its difficult integration with the choice of polynomial multiplication through NWC (only non-binary message).

Transciphering. The transciphering approach has been introduced by Naehrig et al. [59] in 2011. This approach is an essential element of what is called *hybrid homomorphic framework* [60] which proposes a practical approach for homomorphic outsourced computation.

The principle is to use a classical symmetric encryption scheme to upload the sensible data to the homomorphic server. This implies that the upward communication overhead is almost nonexistent (beside the key exchange). The encryption scheme is then transformed from classic to homomorphic directly on the server. This is possible due to the capacity of

[I] The function $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ refers to the Euler's totient function. This function returns, for any natural $m > 0$, the number of integer n mutually prime with m, for all $n \leq m$.

[II] By CRT applied to polynomials.

the homomorphic scheme to evaluate in the encrypted domain the decryption circuit of the classical symmetric scheme.

This approach has been improved by Canteaut et al. [61] to reduce the transciphering latency. They propose to use a lightweight additive IV-based stream cipher as the classical symmetric scheme.

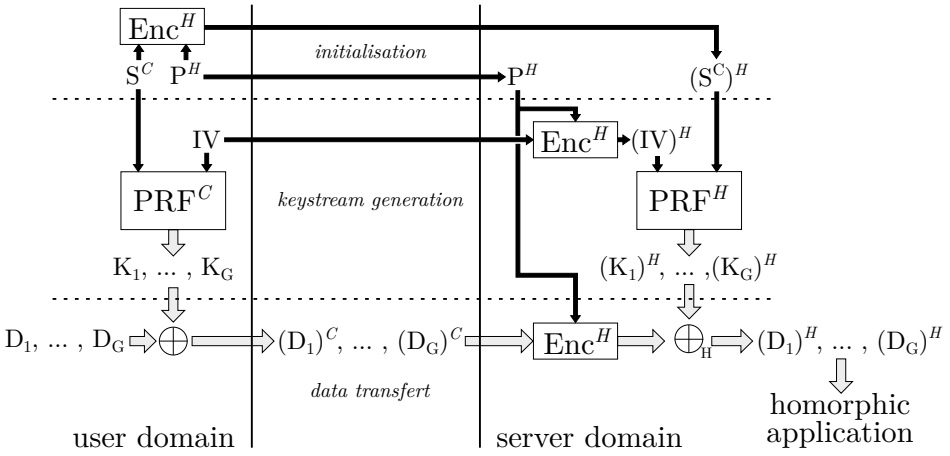


Figure 1.2: Stream cipher based transciphering.

Figure 1.2 presents the principle of their stream-based transciphering protocol. The user and the homomorphic sever agree on a Pseudo-Random Fonction (PRF) and on a homomorphic encryption scheme. The PRF function is used to generate keystream elements: in classical form on the user side, and in homomorphic form on the server side. For this keystream generation, the user only need a classical secret key $(S)^C$ and a set of Initialization Vector IV. On the server side, only the homomorphically encrypted secret $(S^C)^H$ and initialisation vector $(IV)^H$ are required to generate the homomorphic form of the keystream. The data transfer from the user to the server is then a simple classical encryption of the data using the keystream generated on the user side. At reception, the server over-encrypt those data with the homomorphic scheme and apply the homomorphic XOR operation (ciphertext addition) with the homomorphic version of the keystream. The resulting elements are then the homomorphic ciphertexts of the initial data.

As specified in the Figure 1.2, the communication is decomposed in three different phases. First, the *initialization* consists in the choice of the PRF and homomorphic scheme, and in the key exchange. Second, the *keystream generation* that is somehow independent of the concrete transfer of the payload as it can be performed in advance[I]. Third, the *data transfer* is done without any data overhead because the symmetric scheme does not have any. The server has only to transcipher the received data with the homomorphic keystream when they are required for further homomorphic computations.

As a drawback, this approach imposes to chose the homomorphic scheme parameters such that it can evaluate the PRF function plus the application desired by the user. This has a significant impact on L-FHE schemes as they have to dimension their parameters to have a sufficient noise gauge.

[I]From time to time, a new IV may be required to refresh the keystream generation, but this may be considered negligible as the user and server may also agree on an IV generation protocol without compromising the security.

It is important to note also that it only solves the communication overhead for upward communications. Downward communications would require transciphering from homomorphic encryption to classic encryption and that is not possible. Nevertheless, numerous applications would not require large downward communications and the overhead is still reasonable to pay.

1.4.2 Choosing an HE scheme and a plaintext space

The question of concrete choice of homomorphic scheme is of importance for whom want to implement a secure outsourced application. For instance, only a truly FHE scheme is able to address generic cloud computing applications with function not known beforehand. In this case, the fourth generation appears as the most promising. In other scenarios, like a cloud medical storage system, it is reasonable to consider that all the functions to apply on the private data are specified beforehand. In this case, a dedicated L-FHE scheme from second or third generation is enough.

Nevertheless, even when only considering the latter case, each scheme have their practical advantages. A significant example of this is given with the comparison of the FV and SHIELD made by Guillaume Bonnoron during his PhD thesis [9].

Furthermore, even when considering a single scheme, the choice of a plaintext space (modulus t) along with a message encoding method may have a significant impact on the efficiency of the homomorphic evaluation. The second paragraph of this subsection details the current knowledge on this matter.

Comparison of FV and SHIELD. Despite being both constructed upon the RLWE problem, the two schemes are quite different. FV ciphertexts are vectors of polynomials in R_q^2 and SHIELD ciphertexts are matrices of polynomial in $R_q^{N\times 2}$ with $N = 2 \cdot \lceil \log_2(q) \rceil$.

Comparing the schemes noise growth, Bonnoron shows that FV is more suitable than SHIELD for the evaluation of arbitrary binary circuit (bounded by scenario of Figure 1.1a). In that case, both FV and SHIELD parameters need to expand to guarantee correctness and security for larger multiplicative depth. Due to its vector structure, a FV ciphertext is then much smaller than a SHIELD ciphertext.

When considering optimized circuits that require multiplication with fresh ciphertexts (scenario of Figure 1.1b), SHIELD noise growth is far lower tan FV's one. Hence SHIELD parameters (n, q, σ) do not need to expand much to satisfy security and correctness for larger multiplicative depth evaluation capability. For FV, the parameters still have to expand quite significantly to satisfy the required multiplicative depth.

When considering batching as an SIMD computational improvement, the favorable noise growth of a batched SHIELD ciphertext in the case of an optimized circuit does not apply anymore (it is even worse with arbitrary circuits). Thus, batching with SHIELD requires a great expansion of its parameters to ensure correctness and security. Hence, SHIELD becomes quickly non-advantageous compared to FV over which batching does not significantly impact parameter ranges.

This comparative example shows the disparity of HE schemes. Further comparisons of this type are still necessary to have a quantitative classification of the differences and use-cases of the different schemes.

Plaintext space and message encoding choices. For RLWE-based homomorphic schemes, the decryption homomorphism expresses the equivalence between operations on $\mathbb{Z}_q[X]/(\Phi_m(X))$ (ciphertext space) and operations on $\mathbb{Z}_t[X]/(\Phi_m(X))$ (plaintext space). The encrypted application has then to define how its data are encoded over the message space. Both t and the encoding method impact the efficiency of the encrypted computation.

The encoding methods are dependent of the use of the batching optimization. Without batching, a simple encoding of a message consists in setting the message as the coefficient of degree zero of a polynomial in $\mathbb{Z}_t[X]/(\Phi_m(X))$. With batching the encoding consists in setting the r messages as coefficient of degree zero of the r polynomials in $\mathbb{Z}_t[X]/(F_g(X))$ with g in $1, ..., r$. In both examples, the encrypted computations are equivalent to operations in $\mathbb{Z}_t$ (modular arithmetic modulo t).

Concerning the choice of t, handling binary message ($t = 2$) allows the use of comparisons in the ciphertext domain (test of bit-sign, masking, etc.). Thus, it could simplify the homomorphic evaluation for conditional operations. Nevertheless, the use of binary messages requires to decompose the clear application into an equivalent boolean circuit. This may increase the number of encrypted operations to perform. To get a rough idea, consider the boolean circuit of an 8-bit adder rather than being able to perform the addition in one equivalent homomorphic operation ($t = 2^8$).

A recent work from Jäschke and Armknecht [62] specifically addresses the problem of choosing this parameter t. They study this choice with regards to encoding methods for efficient natural, integer and rational arithmetic in the encrypted domain.

Considering the number of operations to perform in the encrypted domain, they show that the optimal choice for t is 2, given the straightforward encoding methods described above. They also consider an encoding in Galois Field[I] $GF(t^k)$ (k in $2, ..., n$ or $2, ..., d$ in case of batching). Nevertheless, it appears to always requires more operations than with the straightforward method.

Considering now the multiplicative depth metric, they observe that small prime values for t may result in shallower depth than $t = 2$, but such choices for t are hardly suitable for generalized encrypted applications. Hence, the authors conclude that the optimal choice for t with respect to the encoding method depends on the application. In this thesis, we consider binary message space ($t = 2$), as it appears to us the most suitable for non-specialized applications.

After having considered the work around the practical usage of homomorphic encryption, we now focus on the implementation of RLWE-based encryption schemes.

1.5 Implementation of RLWE-based schemes

In this section, we are interested in the problems of implementing RLWE-based schemes. We are therefore seeking to position our work within the related works.

1.5.1 Positioning on hardware implementation

Usually, homomorphic encryption schemes are first implemented in software. This allows to test the scheme proposals, and to easily explore algorithmic optimizations.

[I] This encoding takes into account the overall plaintext structure (polynomial of degree less than k with coefficient in $\mathbb{Z}_t$) that is isomorphic to $GF(t^k)$.

For this thesis, it was already a decision to position ourselves on a hardware implementation. This is explained by the performance limitations one experiments with software only solutions for encrypted-computing. For instance, considering the encrypted-computing application on genomic data from Singh et al. [63], the timings are in the order of half an hour for computing the equivalent of 20 thousands logic gates. Thus, existing software implementations have not received any particular attention in this work. However, we mention here some of them that are present in the literature, in case the reader wishes to go further.

To the best of our knowledge, four open software libraries implementing homomorphic schemes based on the RLWE problem are currently available. SEAL [64] from Microsoft Research, PALISADE [65] from the New Jersey Institute of Technology (NJIT), FV-NFLlib [66] from CryptoExpert, and Cingulata [67] from CEA-List. The literature mention other implementations accessible when asking the authors. Among others we mentions the FV-FULL-RNS from Bajard et al. [4] and the SHIELD implementations [8, 68].

During our work, we discuss in particular a variant of FV implemented in the PALISADE library that is proposed by Halevi, Polyakov and Shoup [5]. This variant is a simplification of Bajard et al.'s work [4] making the FV scheme fully compatible with a different representation of number called the Residue Number System (RNS). The reasons and the description of this representation are presented throughout this document.

Before presenting existing hardware implementations related to RLWE-based homomorphic encryption, we first discuss the different implementation issues from an high level point of view.

1.5.2 Hardware implementation issues

The initial problem concerns the large size of the parameters n and q. It is complicated by their high dynamics coming from the variation of multiplicative depth requirements for different encrypted applications. Both problem take their roots from security and correctness requirements as explained in section 1.3. To get an approximate idea, the degree n can reach several thousand, and the modulus q several hundred bits. Consequently, both the computational and memory complexities of the underlying operations are of major importance.

These operations are polynomial multiplications, polynomial reductions, polynomial additions, scale-and-rounds and modular reductions. In practice, polynomial additions are not problematic compared to the others, thus the literature does not mention any specific optimization for them.

The most expensive operation is commonly the multiplication of polynomials. This is due to its quadratic complexity with the degree n considering the schoolbook algorithm. But two other types of algorithms are commonly known to reduce this complexity. The first regroups Karatsuba [69] and Toom-Cook [70] algorithms extended to polynomials. Their complexities are respectively $O(n^{1.585})$ and $O(n^{1.465})$. The second regroups the algorithms based on Number Theoretical Transfrom (NTT) [71] (Fourier transform over a finite-field) with asymptotic complexity in $O(n \log n)$.

The polynomial multiplication is usually followed by a polynomial modular reduction to get back to the considered polynomial ring. The complexity of this polynomial modular reduction is dependent of the hamming weight[I] h of the polynomial modulus, and of course also

of the degree n [72]. It also possible to compute these polynomial reductions by transposing integer reduction algorithms to the polynomials (Barrett reduction for instance). A special case of NTT-based algorithm for polynomial multiplication, called Negative Wrapped Convolution (NWC), allows to directly perform the multiplication in the considered polynomial ring without the necessity of a polynomial reduction. The drawback is that it is only compatible with power of two cyclotomic polynomial rings (i.e. $R = \mathbb{Z}[X]/(X^n + 1)$), and hence it makes the batching of binary messages non-possible.

A scale-and-round operation is applied to each coefficient of a polynomial. Thus, it has a linear complexity with n and a constant complexity dependent of the sizes of the scaling value and coefficients. Usually, this scaling value involves a division by q. Depending on the choice of q, this operation may be made trivial (q a power of two). But some other design choices may also restrict the form of q.

Finally, the coefficient arithmetic is most of the time modulo q. Thus, the choice of an effective modular reduction allows the complexity of higher level operations to be positively influenced. Once again, depending on the choice of q, this operation may be made trivial, or just have a reduced cost.

Now that the general implementation issues for the RLWE schemes have been presented, we will be able to clarify these issues by considering the related works.

1.5.3 Related works on hardware implementation

The intersection of these issues makes difficult a proper comparison of all the works that led to hardware implementations. Numerous approaches at application level may drive the implementation choices. For instance, choosing to use the batching technique imposes to avoid NWC for polynomial multiplications, or choosing a power of two modulus q is incompatible with RNS representation. Furthermore, a good part of the existing works transpose some approaches from non-homomorphic lattice-based cryptography toward the homomorphic context. The resulting hardware will often not be fully compliant with the actual problematic of the HE context.

To take into account this situation, we try to avoid premature comparisons. We have chosen to classify the different works according to their approach to implement polynomial multiplication. However, this level of classification hardly takes into account the other lower level choices that may have motivated the authors. Thus, we will start by mentioning some fairly general techniques to solve these lower-level problems. Then, related works will be described w.r.t. their choices of polynomial multiplications.

Residue Number System. The large modulus q influences the complexity of the basic arithmetic. With its growth for large multiplicative depth, the classic multi-precision arithmetic shows some limitations. It involves a lack of parallelism due to intermediate result propagation leading to important implementation cost and/or low execution performances. Consequently, some work in the literature have proposed a different representation system called the Residue Number System (RNS).

The RNS is a non-positional representation of numbers according to a basis of mutually prime moduli $q_1, ..., q_k$. This representation is a direct consequence of the Chinese Reminder

[I] number of non-zero coefficients

Theorem (CRT) which expresses the ring isomorphism $\mathbb{Z}_q \cong \prod_{1 \leq i \leq k} \mathbb{Z}_{q_i}$. Under this representation, modular arithmetic modulo $q = \prod_{1 \leq i \leq k} q_i$ is performed with k smaller and independent modular operations. For additions, subtractions and multiplications, the RNS representation is an efficient way of creating parallelism, but when it comes to divisions, some more complex computations are required.

Modular arithmetic. With or without RNS representation, the question of computing modular arithmetic arises. For modular additions and modular subtractions, input operands are usually bounded by the considered modulus (q or the q_i's in case of RNS representation). Thus, they require only one addition, one subtraction and one comparison to be performed.

In the case of a modular multiplication, a reduction operation must be performed. Avoiding the use of slow algorithms like Euclid division, the complexity of fast algorithm depends on the considered modulus. In LWE-based cryptography, security does not restrict the choice of q, but its size $\log_2 q$. When there are no additional restrictions due to particular implementation approaches, one may choose q to be a power of two. Hence, the divisions and modular reductions are trivially performed.

Considering RNS representations, the different modulus q_i have to be co-primes. It forces oneself to look for less trivial modular reduction algorithms. One may be interested in some special primes like Mersenne's or more generally Solinas' primes. For these primes, modular reductions are performed with modular additions and shifts.

For NTT-based polynomial multiplications, the modulus q has to be chosen such that the NTT exists. This is often incompatible with special modulus like powers of two or Solinas' primes. In this case, the litterature considers even more generalized modular reduction approaches like Montgomery reduction [73] or Barrett reduction [74].

Polynomial multiplication. The design choices made for the hardware implementation of RLWE-based encryption focus mainly on the objectives of efficient polynomial multiplications.

In the litterature, we found only Mkhinini et al. [75] implementing the schoolbook algorithm. Despite its simple adaptation to any choice of n and q, the implementation is limited in execution performance due to its asymptotic complexity in $O(n^2)$.

Once again, we found only one work that implement the polynomial multiplications using the Karatsuba algorithm, and no work implementing Toom-Cook algorithms. The in-depth work of Migliore et al. [76] shows the advantages and the limitations of the Karatsuba approach. Due to the absence of major constraints in the application of Karatsuba, they are able to finely tune the FV's parameters for the desired security and multiplicative depth requirement. In particular, their approach is compatible with batching of binary messages and they are free to choose q as a power of two. Nonetheless, the complexity of Karatsuba can only compete with NTT-based approaches up to some point in the growth of parameter sets. They identify the performance turning point for ($n = 6144$, $\log_2 q = 512$) compared to the implementation of T. Pöppelmann et al. [77] ($n = 16384$, $\log_2 q = 512$).

Numerous work have chosen NTT-based polynomial multiplication. The main motivation is the reduced asymptotic complexity, but this does not come without a certain complexity of implementation. In a nutshell, a NTT is a Fourier Transform over a finite-field, and requires the existence of a primitive root of unity over this field. Hence, fast algorithms for this transform allows the computation of convolutions in $O(n \log n)$ rather than $O(n^2)$, but this

at the cost of $O(n)$ precomputed values called *twiddle factors*. An attractive variant of NTT-based polynomial multiplication is the previously mentioned Negative Wrapped Convolution (NWC). It allows to perform the convolution on n-points rather than on $2n$ for a classic NTT-based convolution, and it gives the result directly modulo $X^n + 1$. This is advantageous when choosing power of two cyclotomic polynomial rings (more described in Chapter 2). Nevertheless, the complexity of implementing NTT-based approaches in addition to the restrictions for their existence often lead to limited design flexibility.

The most detailed NTT-based implementation we found are those of Pöppelmann et al. [77] and Roy et al. [78]. The former implements a cached-NTT to improve the data locality when computing the NTTs. Due to restrictions on q for the existence of the NTT, they could not choose a modulus being a power of two, hence they consider Solinas primes for efficient modular reduction. They implement only two sets of parameters ($n = 4096$, $\log_2 q = 124$) and ($n = 16384$, $\log_2 q = 512$), and notice the problematic of the large coefficient sizes, in particular for the scale-and-round operations.

The second work, from Roy et al. [78], presents a co-processor implementing building block operations for NTT, RNS representation and scale-and-round operations. In particular they consider the RNS representation to increase the parallelism during NTT-based polynomial multiplications. To reduce the memory complexity, they also choose to store only a subset of the twiddle factors and to compute the others on-the-fly. This implies around 10,000 bubbles in the computation of the NTTs. Their co-processor is designed to handle polynomial with degree up to $n = 2^{15}$ and, as they do not implement the NWC, they choose to implement a Barrett polynomial reduction. Finally, it has to be noted they also have limitations in the calculation of the scale-and-round operations. This is mainly due to the necessity of going back from the RNS representation to a classical multi-precision representation.

Among the other works, Öztürk et al. proposed in 2015 [79] and in 2017 [80] two different versions of an RNS/NTT based accelerator. In both version, they implement iterative NTTs. In the first version they choose to pre-compute the different NTT twiddle factor sets on the host side, and send them along with the polynomial coefficients to their accelerator. In the later version they store all the coefficient inside the BRAMs of the targeted FPGA. In the first case, it involves higher communication cost, and in the second case it involves a large storage cost.

In 2017 Cousins et al. [81] developed data flow NTT as a primitive of a co-processor for SHE applications. The data flow approach for NTTs allows a higher throughput at the cost of duplicated twiddle factors. They chose to store the NTT twiddle factors in ROM filled up at compile time. They note that the storage cost for these twiddles is rather prohibitive.

Finally, other works can be mentioned as variants of those presented above. We have not seen any significant improvement in the latter. Among others: Khairallah et al. [82] implement of a cached-NTT but consider RNS representation contrary to Pöppelmann et al. [77], and Chen et al. [83] with memory access NWC addressing rather small degree $n \in [256; 2048]$ and small modulus $\log_2 q < 57$.

Remarks on related works. Our review of the literature has led us to note the difficulty of proposing a consistent approach with respect to all the implementation issues. Indeed, the most flexible polynomial multiplication approaches allow to simplify scaling and modular reduction operations. But this only pays up to some size of parameter sets.

The NTT-based approaches for polynomial multiplication are more complex to implement. A first issue due to the growth of n is the routing of coefficients required by iterative-NTT

algorithms. This explains the works of Pöppelmann et al. [77] and Khairallah et al. [82] exploring cached-NTT algorithms. These algorithms are designed to take the best from a distributed computing environment.

A second issue is in the handling of twiddle factors. When both n and q get larger, the required storage capacity for these values, or the communication cost for bringing them in, is problematic as one may see in Cousins et al. [81] and Öztürk et al. [79, 80] works. This explain the choice of Roy et al. [78] to compute them on-the-fly.

A third issue is in the handling of scale-and-round operations with NTT-based polynomial multiplications. Indeed, the NTT does not allow simple choice for q to compute these scaling operations (as seen in Pöppelmann et al. [77]). The RNS representation seems to make it worse for this particular case as seen in Roy et al.'s work [78]. This is because they have to get back into a classic multi-precision representation for this particular operation.

With regards to all the different issues, it appears to us more important to look for a flexible acceleration approach than one that is too early optimized for performances. And for this, we will require to fix the case-study to a specific HE scheme. Then we will look to propose an approach that takes into account, as much as possible, all the dimensions of the problem.

1.6 Conclusion and positioning of this thesis

Throughout this chapter we have taken the time to present homomorphic encryption. In particular, we have sought to describe as much as possible the elements that influence hardware acceleration strategy for this new cryptography.

After studying the different generations of encryption, we focused our study on L-FHE schemes based on the RLWE problem. This choice implies that sizing parameters are dependent on the considered encrypted application. Hence, the hardware acceleration approach must take into account the large dynamic of parameters.

This was followed by a presentation of the different application approaches for the practical use of homomorphic cryptography. This introduces to the batching optimization and the transciphering approach that should be considered for proposal of consistent hardware acceleration strategy. In particular, the transciphering involves in practice a minimal multiplicative depth requirement.

Finally, the exploration of related works on the implementation of RLWE-based schemes present the issue of defining an implementation strategy that takes into account all the memory and computational problems. This strategy should be flexible enough to improve the practicability of L-FHE while avoiding too much premature specialization that are not suitable with the growth of parameters.

In order to define such a strategy in the light of the related works, we choose to consider a specific case study. We select for this the FV scheme as a representative of the second generation of HE scheme, being quite well-accepted by the FHE community. Therefore, our work has been focused on the hardware acceleration of FV.

Chapter 2

Definition of an acceleration strategy for the FV scheme

This chapter presents our analysis of the Leveled-FHE scheme of Fan and Vercauteren (FV) towards the definition of a flexible hardware acceleration strategy.

After introducing the FV scheme along with some details on security, correctness and parameterization, a first implementation strategy is set upon the analysis of a performance profiling. The proposed strategy consists in coupling the RNS representation of numbers with NTT-based polynomial ring multiplications. The feasibility of the coupled strategy is then studied, and some concomitant works completing the strategy are reminded. Our analysis of the FV scheme concludes by the orientation of our work toward the exploration of an hybrid computing approach.

2.1 The Fan and Vercauteren (FV) SHE scheme

For completeness, this section describes the RLWE-based L-FHE scheme proposed by Fan and Vercauteren in 2012. In addition to the primitives' presentation, this description is intended to give an idea of the complexity of parameter derivation. These parameters must ensure both security and correctness of the scheme.

The primitives' description is based on the original paper from Fan and Vercauteren [7]. The discussion about corectness is based on the paper of Lepoint et al. [84] which is itself based on Bos et al.'s work [39].

2.1.1 Preliminaries

The motivation of Fan and Vercauteren is to transpose the LWE-based *scale-invariant* scheme of Brakerski [31] to the RLWE problem. They construct their cryptosystem over the scheme from Lyubashevsky et al. [54], presented in the previous chapter (subsection 1.3.2). They essentially adapt the scheme to make it homomorphic by defining addition and multiplication primitives.

They easily construct an addition primitive but things get more complex for multiplication. They define a multiplication primitive in three distinct steps. First, the proper multiplication operation is a tensor product where ciphertexts are seen as vector of R_q^2. The resulting vector is then a non-canonical ciphertext in R_q^3. Second, the non-canonical ciphertext is scaled by

t/q to reduce the noise level. Third, the non-canonical ciphertext is relinearized to get back to a canonical form (vector in R_q^2).

They present two solutions to perform the relinearization operation. Both rely on a "special RLWE sample" called the *relinearization key* masking a part of the secret (i.e. $\mathbf{s}^2$). The first one requires a decomposition of $\mathbf{s}^2$ in a base T. The second involves another modulus g, chosen large enough to efficiently mask $\mathbf{s}^2$.

The following description of FV's primitives is based on a set of parameters that are more or less related to correctness and security. The required level of security is expressed according to a parameter λ representing the minimum number of operations (2^λ) to break the cryptosystem. We remind also that the correctness requirement is expressed according to the multiplicative depth L of the encrypted application. Finally, as introduced in Subsection 1.4.2, the choice of the plaintext modulus t is made depending on the encrypted application.

These requirements involve the derivation of the following parameters:

- n: degree of the cyclotomic polynomials defining the polynomial ring R.
- q: ciphertext modulus defining R_q the ciphertext ring.
- σ: error size of a normal distribution χ over R_q.
- T: (relinearization version 1) decomposition base. For convenience, $l_T = \lfloor \log_T(q) \rfloor$.
- g: (relinearization version 2) relinearization modulus, requiring also an extended error distribution χ' over R of standard deviation σ_g.

2.1.2 Cryptosystem primitives

To avoid cluttering the reader with the mathematical definitions of FV primitives, we describe here only their functionality. Details are available in the appendix of this chapter in Section 2.6.

Core primitives. The core primitives define the generation of the private and public keys, and the encryption and decryption processes.

- FV.SecretKeyGen(λ): provides the private key $\mathtt{sk} \in R_2$.
- FV.PublicKeyGen($\mathtt{sk}$): provides the public key $\mathtt{pk} \in R_q^2$.
- FV.RelinKeyGen($\mathtt{sk}$, T or g): provides the relinearization key $\mathtt{rlk} \in (\ R_q^{l_T}$ or $R_{gq}^2)$.
- FV.Encrypt($\mathtt{pk}$, $\mathbf{m}$): encrypts a message $\mathbf{m} \in R_t$ in a ciphertext $\mathtt{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in R_q^2$.
- FV.Decrypt($\mathtt{sk}$, $\mathtt{ct}$): decrypts a ciphertext $\mathtt{ct}$ to retrieve the encrypted message $\mathbf{m}$.

Evaluation primitives. The evaluation primitives are the basic operations of encrypted computing. These are the ones that are of particular interest to us in this thesis.

- FV.Add($\mathtt{ct}_a$, $\mathtt{ct}_b$): performs an encrypted addition equivalent to an addition over R_t.
- FV.Mul($\mathtt{ct}_a$, $\mathtt{ct}_b$): performs an encrypted multiplication equivalent to a multiplication over R_t. This primitive performs both the tensor product and the scaling by t/q to reduce the noise. The result is a non-canonical ciphertext $\tilde{\mathtt{ct}} = (\tilde{\mathbf{c}}_0, \tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2) \in R_q^3$.
- FV.Relin($\tilde{\mathtt{ct}}$, $\mathtt{rlk}$): relinearizes a ciphertext to get back to a canonical form $\mathtt{ct} \in R_q^2$.
- FV.Mul&Relin($\mathtt{ct}_a$, $\mathtt{ct}_b$, $\mathtt{rlk}$): performs the multiplication and the relinearization in a single flow.

The efficiency of FV primitives is mainly dependent on the parameters of the scheme. The derivation of these parameters is quite complex and relies on security and correctness requirements. In order to present the parameterization process, security assumptions and correctness requirements are presented in the following subsections. It is then followed by examples of parameter sets to get a fair idea of the parameter ranges.

2.1.3 Security assumptions

In this subsection are presented security matters for completeness of the presentation. In particular, the *weak circular security* assumption is required to construct the relinearization primitive, and the *binary secret* assumption reduces the noise in ciphertexts. Both assumption are essential for the efficiency of the scheme.

Decision RLWE. The FV scheme relies on the decision version of the RLWE problem introduced in the previous chapter. At the time of their initial work, the proof of hardness reduction of decision-RLWE was based only on cyclotomic number-fields. Consequently, the FV definition considers cyclotomic polynomial rings, which are $R = \mathbb{Z}[X]/(F(X))$ with $F(X) = \Phi_m(X)$ being a cyclotomic polynomial. About the modulus q, they had the knowledge that hardness does not rely on a special shape for q [85], but on its size with respect to n and σ.

A legitimate question is to know if the results of Peikert et al. [56] could be applicable to the FV definition, and if it would change its structure somehow. It could makes us consider other polynomial rings that may be more convenient for practical implementations. It seems that it applies without any structural consequences. Nevertheless, as a precaution, this work won't rely on this assumption. Hence, it continues to consider FV with cyclotomic polynomial rings only.

Weak circular security. The particular step of relinearization transforms a non-canonical ciphertext to its canonical form. This particular step requires a partial knowledge of the square of the secret (i.e. $\mathbf{s}^2$). Indeed, both versions of relinearization rely on masked versions of $\mathbf{s}^2$ that are added as a special error term into a classical RLWE sample. They make the assumption that the scheme is still secure knowing this special RLWE sample called *relinearization key*.

Binary secret. In order to reduce the noise in ciphertext, Fan and Vercauteren rely on a security result from LWE settings and they make the assumption that it carries over RLWE settings. Namely, that the secret and the noise elements for encryption could be sampled from R_2 instead of R_q. For the secret, they actually take it with a low Hamming weight h (number of non-zero coefficient) to reduce the private key size, while still ensuring sufficient entropy.

2.1.4 Correctness w.r.t. noise growth

In this subsection are presented correctness matters for the completeness of the presentation. In particular, we seek to highlight the influence of the parameters on the noise and its growth. This is important to get a proper understanding on the complexity of FV parameterization.

For the following descriptions, we will consider these notations. We note a ciphertext and its polynomials $\mathtt{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in R_q^2$. To distinguish different level of noise for a ciphertext, we

will note $\mathtt{ct}^{(l)}$ a ciphertext at level l for the considered operation (addition or multiplication). The level is defined w.r.t. a binary tree of operations as presented in Chapter 1. We set $\Delta = \lfloor \frac{q}{t} \rfloor$, and we note $r_t(q) = t(q/t - \Delta)$. Finally, there are two types of sampling in our discussion. When we refer to a particular distribution (e.g. χ), it is considered a sampling according to this distribution. In other cases, it is considered a uniform distribution over the concerned set.

Basic notions. By definition, the noise of a ciphertext encrypting a plaintext element $\mathbf{m} \in R_t$ is the polynomial $\mathbf{v}$ such that $[\mathbf{c}_0+\mathbf{c}_1\mathbf{s}]_q = [\Delta[\mathbf{m}]_t+\mathbf{v}]_q$. For a more in-depth understanding of the definition, please refer to the details of the FV primitives in the Section 2.6. Note also that the decryption function erases the noise when scaling by t/q and rounding to the nearest integer. The final reduction modulo t gives back the message.

In order for the decryption to be correct, the noise $\mathbf{v}$ must not grow past a certain threshold: $\|\mathbf{v}\|_\infty < (\Delta - r_t(q))/2$. In practice, one is interested in upper-bounding the noise growth to easily express the correctness requirement w.r.t. the application. Hence, in order to choose the parameters of the scheme, the initial noise and its growth along homomorphic operations have to be known.

Beforehand, here are some important constants involved in the noise equations:

- B_{key}: the infinite norm's upper bound of an element sampled from the uniform distribution over R_2. In this case, $B_{key} = 1$.
- B_{err}: the infinite norm's upper bound of an element sampled from the χ distribution over R_q. This value is usually taken as $B_{err} = 6\sigma$.
- δ: the expansion factor of the considered ring R. By definition, $\delta = \sup_{\mathbf{a},\mathbf{b} \in R} \left\{ \frac{\|\mathbf{ab}\|}{\|\mathbf{a}\|\|\mathbf{b}\|} \right\}$, and express the maximum norm's expansion during a product in R. When considering the Euclidean norm $\|\cdot\|$, δ is bounded by $\sqrt{n}$. Numerous work consider the infinite norm $\|\cdot\|_\infty$ for convenience, and hence bound δ by n, with n being the dimension of the ring.

  In our case, the handled polynomial are actually sampled accordingly to some probability distribution, and so tighter bounds on δ may be found. For example, Halevi et al. in [5] take $\delta = 2\sqrt{n}$ on experimental grounds. As it is not the purpose here to refine this upper bound, $\delta = n$ is considered as a first acceptable approximation.

After these preliminaries, the *initial noise*, the *additive noise* and the *multiplicative noise* are studied.

Initial noise. The initial noise is the one of a freshly encrypted ciphertext. For studying the correctness, an upper bound has to be considered. It is reminded that the public key is constructed from the secret polynomial $\mathbf{s}$ sampled from R_2, a polynomial $\mathbf{a}$ sampled from R_q and an polynomial $\mathbf{e}$ sampled from χ: $\mathtt{pk} = ([-(\mathbf{as} + \mathbf{e})]_q, \mathbf{a})$.

Furthermore, when encrypting a message $\mathbf{m}$, a polynomial $\mathbf{u}$ is sampled from R_2 and two polynomials $\mathbf{e}_1$ and $\mathbf{e}_2$ are sampled from χ. Hence, when evaluating the fresh ciphertext with the secret $\mathbf{s}$ to highlight the noise, the following equation is derived.

$$\begin{aligned} \mathbf{c}_0 + \mathbf{c}_1\mathbf{s} &= \Delta\mathbf{m} - (\mathbf{as} + \mathbf{e})\mathbf{u} + \mathbf{e}_1 + (\mathbf{au} + \mathbf{e}_2)\mathbf{s} \\ &= \Delta\mathbf{m} - \mathbf{eu} + \mathbf{e}_1 + \mathbf{e}_2\mathbf{s} \\ &= \Delta\mathbf{m} + \mathbf{v}_{init} \end{aligned}$$

Hence, $\|\mathbf{v}_{init}\|_\infty = \| - \mathbf{eu} + \mathbf{e}_1 + \mathbf{e}_2\mathbf{s}\|_\infty < B_{err}(1 + 2\delta B_{key})$. We note this bound B_0 and according to the considered approximations for δ and B_{err} it is roughly $6\sigma(1 + 2n)$.

Additive noise. The addition of two ciphertexts $\mathtt{ct}_a = (\mathbf{a}_0, \mathbf{a}_1)$ and $\mathtt{ct}_b = (\mathbf{b}_0, \mathbf{b}_1)$ gives $\mathtt{ct}_{add} = \left([\mathbf{a}_0 + \mathbf{b}_0]_q, [\mathbf{a}_1 + \mathbf{b}_1]_q\right)$. Considering the addition of two fresh ciphertexts, the noise level of the operands is bounded by B_0. Evaluating the resulting ciphertext ct_{add} with the secret $\mathbf{s}$ to highlight the noise gives the following upper bound:

$$\left\|\mathbf{v}_{add}^{(1)}\right\|_\infty < 2B_0 + r_t(q).$$

Hence, in a case of a binary tree of ciphertext additions, the noise level at depth L is upper bounded by:

$$\left\|\mathbf{v}_{add}^{(L)}\right\|_\infty < 2^L \left(B_0 + r_t(q)\right) - r_t(q).$$

We note this bound $B_{add}^{(L)}$ and according to the considered approximations and the definition of B_0 above, it is roughly $2^L(6\sigma(2n+1)+r_t(q))-r_t(q)$. The impact of the scheme parameters (n, q, σ) over the additive noise level is relatively small.

Multiplicative noise. The noise growth after a ciphertext multiplication is slightly more complex to upper bound. The presented bound is extracted from Lepoint et al. [84] which is itself based on the work from Bos et al. [39]. We consider here only the first version of the relinearization.

Considering the multiplication of two fresh ciphertexts, it can be shown that:

$$\left\|\mathbf{v}_{mult}^{(1)}\right\|_\infty < \delta t(4 + \delta B_{key})B_0 + \delta^2 B_{key}(B_{key} + t^2) + \delta l_T T B_{err}.$$

It is reminded that $l_T = \lfloor \log_T q \rfloor$ and T is the decomposition basis for relinearization. In the case of a binary tree of ciphertext multiplications, the noise level at depth L is upper bounded by:

$$\left\|\mathbf{v}_{mult}^{(L)}\right\|_\infty < C_1^L B_0 + L C_1^{L-1} C_2,$$
$$\text{where } C_1 = \delta t(4 + \delta B_{key}) \text{ and } C_2 = \delta^2 B_{key}(B_{key} + t^2) + \delta l_T T B_{err}.$$

We note this bound $B_{mul}^{(L)}$ and, according to the considered approximations, C_1 is roughly $t(n^2 + 4n)$ and C_2 roughly $n^2(1 + t^2) + 6nl_T T\sigma$.

Contrary to additions, the noise at multiplicative depth L is highly dependent on the scheme parameters (n, q, σ). In particular, the dominant term is in $O(n^{2L+3/2})$. Note that the refinement of δ on experimental grounds may help to not oversize the parameters. For instance, in Halevi et al.'s work [5] the multiplicative noise bound is actually $O(n^{L+1})$.

As the multiplicative noise is tremendously larger than the additive noise, the noise gauge of ciphertexts is usually derived from the multiplicative depth requirements. This is a fair approximation for general studies. Nevertheless, when one knows the concrete application to be performed in the encrypted domain, the noise gauge may be refined taking into account the concrete flow of homomorphic operations. From now on, we will consider the multiplicative depth L to be the generic requirement for homomorphic evaluation capability.

2.1.5 FV parameter sets

In previous subsections, security and correctness notions have been introduced. The discussion now proceeds to the concrete choice of parameters. The objective is to get a fair idea of the sizing parameters in order to set up a consistent acceleration strategy.

Parameter derivation aims to select the smallest parameter set (n, q, σ) that respects security and correctness requirements. For security, the literature mainly relies on estimation model for each known attack against LWE. The state-of-the-art estimator of LWE's security is maintained by Albrecht [53]. For correctness, each designer relies on the correctness equations detailed in previous subsection. This is done with respect to the application/implementation-related parameters (L, t, T ...).

Algorithm 1 FV's parameters selection extracted from [9].
Determines (n, q, σ) according to (λ, L, t, T)

```
1: function ChooseParams(λ, L, t, T)
2:     n = 0
3:     do
4:         n = n + 1
5:         σ = 2√n
6:         q = MinModulus(λ, L, t, T)
7:     while SecEstimation(n, q, σ) > λ
8:     return (n, q, σ)
9: end function
```

Algorithm 1 presents a simple method to choose FV's parameter sets. This algorithm is extracted from Bonnoron's PhD thesis [9] and relies on two functions: `SecEstimation` and `MinModulus`. `SecEstimation` uses Albrecht estimator [53] and should be updated with every improvement of the underlying attack models. The choice $\sigma = 2\sqrt{n}$ is motivated for security by the paper of Lyubashevsky et al. [54]. `MinModulus` simply returns the minimal modulus for which the correctness equation for multiplicative depth L is verified.

Table 2.1: Table extracted from Bonnoron PhD thesis [9]. FV parameters (n–$\log_2 q$) obtained with Algorithm 1. Security estimation for the LWE instances according to the LWE-estimator [53] `commit 61ac716`. Other parameters: $\sigma = 2\sqrt{n}$, $t = 2$ and considering $\delta = n$.

L	$\lambda = 80$ bits		$\lambda = 128$ bits	
	$T = 2^{32}$	$T = 2^{64}$	$T = 2^{32}$	$T = 2^{64}$
1	(1,188–54)	(1,982–87)	(1,878–55)	(3,106–88)
5	(3,711–159)	(4,507–193)	(6,014–166)	(7,292–200)
10	(7,120–303)	(7,917–337)	(11,625–317)	(12,844–351)
15	(10,715–454)	(11,549–489)	(17,507–475)	(18,729–509)
20	(14,405–611)	(15,187–645)	(23,491–639)	(24,755–673)

As an example, Table 2.1 presents some parameters extracted from Bonnoron's PhD thesis obtained with the method described in Algorithm 1. The version of the LWE-estimator used for the `SecEstimation` function is referenced by its commit number (`commit 61ac716`).

The parameters are quite large and they involve basic polynomial elements of important

sizes. A polynomial in R_q is 8kB for the smallest parameter set and 2MB for the largest. It is noticeable that for almost all parameter sets, the modulus q does not fit into a single-word with regards to standard computing architectures.

Regarding element sizes, an FV ciphertext is at least 16kB and up to 4MB for the considered parameters (to encrypt a 1-bit message). A private key is simply a polynomial in R_2, and hence could fit in 3kB for the largest considered setting. The public elements composed of the public key and the relinearization key represent 48kB up to 50MB of data, with the relinearization key of 32kB and 46MB respectively.

2.1.6 Concluding remarks

In this section, the FV scheme has been presented with some in-depth on security and correctness in order to understand the derivation of FV parameters. From this overview, the general problematic of L-FHE scheme is understandable. In order to evaluate large applications in the encrypted domain, the sizing parameters increase significantly. And in addition to this issue, the on-going refinement of LWE and RLWE parameters for security does not allow us to focus the implementation efforts on particular parameter sets.

After the presentation of the scheme and its parameters, the next step is to identify and quantify the performance bottlenecks of homomorphic evaluation with FV. The next section presents the profiling of a typical homomorphic evaluation.

2.2 Profiling and hardware implementation strategy

A straightforward application to evaluate in the encrypted domain is a transciphering keystream generation. As seen in the previous section, a FV's ciphertext is at least 10^4 times larger than the data it encrypts. Hence, the transciphering protocol described in subsection 1.4.1 appears mandatory for a generalized usage of homomorphic encryption.

In this context, the generation of homomorphic keystream elements becomes a major computational workload on the server side, which is somehow independent of the user's application. Consequently, we choose to quantify the performance bottlenecks of the FV primitives with a homomorphic keystream generation.

2.2.1 Experimental description

The implementation of the FV scheme from Carpov et al. [86] is used in our experiment. They implement the scheme with the second version of the relinearization primitive. This implementation is based on GMP [87] for multi-precision arithmetic and FLINT [88] for polynomial arithmetic.

The choice of FV parameters is controlled by the Cingulata library [67] which contains the scheme implementation. At the time of the experimentation, the derivation rules followed FV's initial paper [7]. Consequently, the parameters are subject to security caution for the following reason. The initial derivation strategy proposed by Fan and Vercauteren is only based on Linder and Peikert analysis of the LWE security from 2011 [89]. Hence, the derivation does not take into account Albrecht work on practical hardness of the LWE from 2015 [90] and 2017 [91]. Nevertheless, the size of the parameters is sufficient to give an overview of the performance bottlenecks.

The choice of the Pseudo-Random Fonction (PRF) for transciphering follows the initial proposition of Canteaut et al. [61] when they introduced stream-based transciphering (i.e.

Trivium [92]). We briefly describe the Trivium cipher for a better understanding of the profiling results.

Trivium has a 288-bit internal state. At initialization, the 80-bit key and a 80-bit Initialisation Vector (IV) are stored in this internal state at specific locations. At each cycle, the cipher performs a step which consists in one shift of the 288-bit shift register, and in 11 XOR and 3 AND of some specific bits.

During a first phase lasting for 1152 steps, the key and the IV are shuffled into the 288-bit register. After this warm-up phase, an pseudo-random bit is outputted at each cycle.

Because Trivium multiplicative depth (AND operations) increases with the number of steps performed, the number of homomorphic keystream elements generated per warm-up is limited. In our case, we generate 57 keystream elements with a noise level $L = 12$ for each warm-up.

The profiling of the generation of 57 homomorphic keystream elements is performed using the Valgrind tool suite [93].

Table 2.2: Profiling results of homomorphic evaluation of Trivium-12. FV implementation from Carpov et al. [86]. $\lambda < 80$, $L = 19$, $n = 8192$, $t = 2$, $\log_2 q = 913$, $\log_2 \sigma = 383$, $\log_2 g = 2739$, , $\log_2 \sigma_g = 783$. Valgrind 3.10 on Intel Core i7-3770.

HE Evaluation	# calls	Cycles ($\times 10^6$)	% Parent	% Total
H Trivium	1	**26,846,959**	**100 %**	**100 %**
+ WarmUp	1	**25,576,914**	**95.27 %**	**95.27 %**
++ FV.Mul&Relin	3,456	25,420,474	99.39 %	94.69 %
+++ FV.Mul	13,824	10,860,382	42.72 %	40.45 %
+++ FV.Relin	3,456	13,577,587	53.41 %	50.57 %
+++ Others	62,208	982,505	3.87 %	3.66 %
++ FV.Add	10,368	134,693	0.52 %	0.50 %
++ Others	31,105	21,747	0.09 %	0.08 %
+ KSGen	1	**1,270,045**	**4.73 %**	**4.73 %**
++ FV.Mul&Relin	171	1,258,445	99.09 %	4,69 %
+++ FV.Mul	684	537,532	42.71 %	2.00 %
+++ FV.Relin	171	672,219	53.42 %	2.50 %
+++ Others	3,078	48,694	3.87 %	0.18 %
++ FV.Add	798	10,366	0.82 %	0.04 %
++ Others	1,526	1,234	0.10 %	0.00 %

2.2.2 Profiling results

The results of the profiling are detailed in Table 2.2. Without surprises, the warm-up phase is far more expensive than actual generation of homomorphic keystream elements as it requires 1152 warm-up steps to generate only 57 useful elements. If one increases the number of useful elements generated to make the warm-up profitable, it results in less noise gauge left for the concrete encrypted application. This highlights the need for an adequate Pseudo-Random

Function (PRF) for SHE transciphering protocol. This is in particular addressed in Méaux PhD. thesis [60].

Our focus is more on the primitives of FV. In both warm-up and keystream generation parts, the FV.Mul&Relin primitive is the principal performance bottleneck, even with three times less calls than for FV.Add primitive. The workload of FV.Relin is also significantly heavier than the workload of FV.Mul (53.4% Vs 42.7%). Note that FV.Relin is supposed to have twice less polynomial multiplications to perform in the second version of the relinearization (see Section 2.6). The difference is mainly due to the coefficients of the polynomials manipulated during relinearization being four times larger than those manipulated during FV.Mul. This highlights the impact of large coefficients on performance bottlenecks.

Digging a bit more into these two steps, they both rely on polynomial multiplications. The underlying algorithms are dependent on the chosen library for polynomial arithmetic. In our case, the FLINT library calls two different approaches: one based on FFT convolution (similar to Schönhage-Strassen algorithm for large integer multiplication), and the other based on Kronecker substitution (basically reduces the problem of polynomial multiplication to a large integer multiplication). The concrete choice of algorithm during computation depends on the FLINT library's internal metrics. Nevertheless, polynomial multiplications represent 87.41 % of the total estimated cycles of the overall encrypted Trivium execution.

Our profiling corroborates the literature's orientations of improving ciphertext multiplications. In particular, the underlying polynomial multiplications have a prominent influence on the performance bottlenecks. Even with an highly optimized library like FLINT, these polynomial multiplications are problematic sue to large degree n. This highlights the requirement of accelerating polynomial multiplications in this context.

It is also noticeable that the coefficient size has a significant impact on polynomial multiplication performances, even with highly optimized library like GMP. This information raises the need for a strategy to manage large coefficients (modulus q).

In addition to this profiling analysis, we have already mentioned that FV parameters must grow in order to evaluate more complex applications. Hence, the need of an hardware approach that scales-up with the parameters is added to those previously expressed.

It appears to us more important to look for a flexible acceleration approach than for one that is too early optimized for specific parameters. In the following subsection, the existing strategies to address the highlighted implementation problematic are discussed. It results in a first definition of our hardware implementation strategy.

2.2.3 Analysis w.r.t. existing implementation strategies

The challenge in implementing the FV scheme – and more generally for RLWE-based HE schemes – is to address both the complexity brought by large modulus q and the complexity brought by large degree n.

In subsection 1.5.3, the Residue Number System (RNS) has been mentioned to tackle large multi-precision arithmetic due to large q. Indeed, multi-precision arithmetic is limited due to intermediate result propagation. This implies difficulties to exploit parallelism leading to important implementation costs or low execution performances. The RNS brings straightforward parallelism and allows a designer to fix himself the size of the RNS basis elements. Hence, the main limitations of multi-precision in our context are theoretically non-existent with RNS.

Subsection 1.5.3 introduces also the existing approaches for polynomial ring multiplications. A quick summary is given here as a reminder. The naive approach to compute polynomial multiplications appears not suitable for homomorphic encryption due to the quadratic complexity over their degree $O(n^2)$.

The Karatsuba and Toom-Cook polynomial multiplications have reduced asymptotic complexities ($O(n^{1.585})$ and $O(n^{1.465})$) compared to the naive approach. They maintain a large flexibility in the choice of the polynomial ring R and thus allow batching of binary plaintexts.

NTT-based approaches have a more important constant complexity, which makes them rather costly to implement. Nevertheless, they have the best known asymptotic complexities $O(n \log n)$ to date. NTT-based approaches are more restrictive regarding FV's parameterization. For instance, the Negative Wrapped Convolution (NWC) reduces the choice of polynomial ring, and is not compatible with batching of binary plaintexts.

Considering the need for flexibility over FV's parameters, Karatsuba or Toom-Cook seem straightforward choices. Nevertheless, this flexibility comes at the cost of limited asymptotic performances. Indeed, the in-depth work of Migliore et al. [76] identify this limitation. From this point of view, our choice would rather be to explore NTT-based approaches.

What finally makes us decide for the latter is the consideration that NTT architectures are really close to those for Discrete Fourier Transform (DFT). Hence, the lack of flexibility of NTT is somehow compensated by years of research from the data signal processing community on efficient DFT architectures. This is not the case for Karatsuba/Toom-Cook approaches that are mainly used at software level.

Our choice is then to improve the performance of the FV scheme considering RNS representation and NTT-based polynomial multiplications. Our goal of flexibility is defined with respect to the ability of accelerating FV's primitives for a large range of parameter sets.

The next section makes a more formal description of the RNS and the NTT-based polynomial multiplications considered in the context of FV. It is followed by the validation of the theoretical feasibility of the RNS/NTT coupled approach for very large parameters.

2.3 Exploration of the RNS/NTT coupled approach

2.3.1 Simplified arithmetic through RNS

The Residue Number System (RNS) is a non-positional representation of numbers according to a basis of mutually prime moduli $\mathcal{B} = q_1, ..., q_k$. This representation is a direct consequence of the Chinese Remainder Theorem (CRT) which expresses the ring isomorphism $\mathbb{Z}_q \cong \prod_{1 \leq i \leq k} \mathbb{Z}_{q_i}$. The particular terminology calls the $\mathbb{Z}_{q_i}$'s the *RNS channels*.

The RNS representation of an element $x \in \mathbb{Z}_q$ is simply obtained by computing its residues modulo each element of the RNS basis $\mathcal{B}$, and by concatenating them into a set of residue $\{x_i\}_{q_i \in \mathcal{B}}$, with $x_i = [x]_{q_i}$. The CRT gives the transformation back from RNS representation to classical representation. Its expression requires some additional notations. For all q_i in the RNS basis $\mathcal{B}$, we denote $q_i^* = q/q_i \in \mathbb{Z}$ and $\tilde{q_i} = (q_i^*)^{-1} \in \mathbb{Z}_{q_i}$. Put in another way, q_i^* is the product of all the elements of $\mathcal{B}$ beside q_i, and $\tilde{q_i}$ is the multiplicative inverse of q_i^* in $\mathbb{Z}_{q_i}$. Hence, the reconstruction of $x \in \mathbb{Z}_q$ from the x_i's is done as follow:

$$x = \left[\sum_{q_i \in \mathcal{B}} x_i \cdot \tilde{q_i} \cdot q_i^* \right]_q . \tag{2.1}$$

Under RNS representation, modular arithmetic modulo $q = \prod_{1 \leq i \leq k} q_i$ is performed with k small and independent modular operations. This essentially stands for additions and multiplications, but regarding divisions things get trickier.

For two numbers x and y in $\mathbb{Z}_q$, their RNS representation according to the basis $\mathcal{B}$ is noted $(x_1, ..., x_k)$ and $(y_1, ..., y_k)$. The computation of $x + y$ in $\mathbb{Z}_q$ is simply the additions of the residues in each RNS channel:

$$[x + y]_q \Longleftrightarrow ([x_1 + y_1]_{q_1}, ..., [x_k + y_k]_{q_k}).$$

Similarly, $x \times y$ in $\mathbb{Z}_q$ is then computed with the multiplications of the residues in each RNS channel:

$$[x \times y]_q \Longleftrightarrow ([x_1 \times y_1]_{q_1}, ..., [x_k \times y_k]_{q_k}).$$

Division is straightforwardly performed in RNS under the following conditions: x has to be a multiple of y, and y has a multiplicative inverse in $\mathbb{Z}_q$. Expressed differently, $x \mod y = 0$, and y is mutually prime with q. Hence the RNS representation of x/y is:

$$[x/y]_q \Longleftrightarrow ([x_1 \times y_1^{-1}]_{q_1}, ..., [x_k \times y_k^{-1}]_{q_k}),$$

where y_i^{-1} is the multiplicative inverse of y_i in $\mathbb{Z}_i$.

When the conditions are not met, it is required to get more information about the division operands. A possible solution is to get back into a positional representation system like classic multi-precision or MRS (Mixed Radix System) [94]. The inconvenient is that changing the representation is rather costly as it can be seen in Roy et al.'s work [78].

In our context, the RNS representation is straightforwardly adapted to the polynomials used in the FV cryptosystem. The RNS representation of a polynomial $\mathbf{a}$ in R_q is simply the concatenation of the polynomials $\mathbf{a}_i$ in the R_{q_i}'s ($q_i \in \mathcal{B}$). The coefficients of the polynomial $\mathbf{a}_i$ being the residues modulo q_i of the coefficients of the polynomial $\mathbf{a}$.

For all primitives requiring only polynomial additions and polynomial multiplications, the adaptation to RNS representation is straightforward. Due to scale-and-round operations, FV.Decrypt and FV.Mul need a special adaptation to make them fully compatible with the RNS.

In the literature, two works have proposed the adaptation of these primitives to RNS while avoiding expensive transformations to and from a positional representation. The first is from Bajard, Eynard, Hasan and Zucca [4] in 2016, and the second is from Halevi, Polyakov and Shoup [5] in 2018. Here we are just interested in the feasibility of a full RNS variant for FV, but it is nevertheless important to note that it is not for free. This will be detailed in Section 2.4.

Most of the polynomial arithmetic of original FV stands in R_q, so the coefficient range is bounded by q. The only exception is during FV.Mul where the tensor product of the two ciphertexts is performed in R. This is due to the way the noise is scaled down by the scale-and-round operation after the tensor product. Consequently, the values taken by the resulting polynomial's coefficients are bounded by $2\delta q^2$. From this, we can express the requirements in terms of RNS basis to compute the primitives of the full RNS variant of FV.

For operations over R_q, one has to select a RNS basis $\mathcal{B} = (q_i)_{i=1}^k$ and set $q = \prod_{i=1}^k q_i$ such that q has the appropriate size for security and correctness. Then, the operations over R_q are performed through k independent operations over the R_{q_i}'s.

For the tensor product over R, one may simply choose an additional basis $\mathcal{B}' = (p_j)_{j=1}^{k'}$ such that $\prod_{i=1}^{k} q_i \times \prod_{j=1}^{k'} p_j$ is strictly larger than $2\delta q^2$. This implies that $p = \prod_{j=1}^{k'} p_j$ must be strictly larger than $2\delta q$. Then, the multiplication over R requires the basis extension from RNS basis $\mathcal{B}$ to the basis $\mathcal{B} \cup \mathcal{B}'$. Once we have the representation according to the latter basis, the polynomial multiplication is performed through $K = k + k'$ independent multiplications over the R_{q_i}'s and R_{p_j}'s. The special operation of basis extension will be expanded upon in section 2.4.

The RNS representation does not improve the complexity of the polynomial multiplication, it just bounds the size of the arithmetic performed in each RNS channel while bringing in parallelism with respect to the basis sizes. The next subsection details the NTT-based approaches to compute the polynomial ring multiplications.

2.3.2 NTT-based polynomial ring multiplications in RNS

The Number Theoretical Transform (NTT) [71] is comparable to a Fourier transform, but stands over a Galois Field $\mathbb{F} = \mathrm{GF}(p^d)$. Let us consider a n-sequence in $\mathbb{F}$ noted $\mathbf{a} = (a_i)_{i=0}^{n-1}$. For n a divisor of $p^d - 1$ and ω an element of order n in the multiplicative group $\mathbb{F}^*$, the transform of $\mathbf{a}$ noted $\mathbf{A} = (A_i)_{i=0}^{n-1}$ is given by:

$$\text{For all } i \text{ in } \{0, ..., n-1\},\ A_i = \sum_{j=0}^{n-1} a_j \omega^{ij}. \tag{2.2}$$

Note that additions and multiplications are those of the Galois field $\mathbb{F}$. The inverse transform of the sequence $\mathbf{A}$ is given by:

$$\text{For all } i \text{ in } \{0, ..., n-1\},\ a_i = n^{-1} \sum_{j=0}^{n-1} A_j \omega^{-ij}. \tag{2.3}$$

With n^{-1} (respectively ω^{-ij}) being the multiplicative inverse of n (respectively ω^{ij}) over $\mathbb{F}$. A multiplicative inverse of an element g in $\mathbb{F}$ is the element g^{-1} such that $g \cdot g^{-1} = 1$ according to $\mathbb{F}$ multiplication. The elements ω^{ij} and ω^{-ij} are called *twiddle factors* in this work.

Both the NTT and its inverse are efficiently computed using Fast Fourier Transform (FFT) approaches. The resulting asymptotic complexity is in $O(n \log n)$. Furthermore, the NTT shares convolution properties with the FFT. It results in the definition of efficient approaches for polynomial multiplications over finite-fields.

In the case of FV and more generally in the case of RLWE-based cryptosystems, it is required to perform multiplication over $R = \mathbb{Z}[X]/(F(X))$ and/or $R_q = \mathbb{Z}_q[X]/(F(X))$. Using a RNS representation of the polynomial's coefficients according to some basis of co-prime moduli $\{q_i\}_{i=1}^{K}$, both multiplications over R and R_q are performed through multiple multiplications over the R_{q_i}'s. To adapt the above NTT definition to this context, one has to consider finite-fields $\mathbb{F}_i = GF(q_i)$, isomorphic to $\mathbb{Z}_{q_i} = \mathbb{Z}/q_i\mathbb{Z}$, over which n-points NTT are defined.

For further discussions, a n-point NTT (resp. INTT) over $\mathbb{F}_i$ is noted $\mathrm{NTT}_{n,i}$ (resp. $\mathrm{INTT}_{n,i}$). We note ω_i an element of order n in the multiplicative group $\mathbb{F}_i^*$. It exists if and only if $n = 1 \mod q_i$ (n is a divisor of $q_i - 1$).

Without taking into account the polynomial modular reduction to get the result in R_{q_i}, let us consider polynomial multiplication over $\mathbb{F}_i[X]$. Let $\mathbf{a}_i$ and $\mathbf{b}_i$ be polynomials in $\mathbb{F}_i[X]$, both of degree n, and $N \geq 2n$ such that N divides $q_i - 1$. With $\odot$ being point-wise multiplications of same size's vector, the polynomial product $\mathbf{c}_i = \mathbf{a}_i \cdot \mathbf{b}_i$ is computed as follow:

$$\mathbf{c}_i = \mathrm{INTT}_{N,i}\left(\mathrm{NTT}_{N,i}(\mathbf{a}_i) \odot \mathrm{NTT}_{N,i}(\mathbf{b}_i)\right). \tag{2.4}$$

For further comparisons, this approach is called Padded Convolution (PC). In this case, one has to pad with zeros the two n-sequence $\mathbf{a}_i$ and $\mathbf{b}_i$. Hence, the product $\mathbf{c}_i$ is exactly of degree $2n$. A polynomial modular reduction modulo $F(X)$ is then performed to get the result in R_{q_i}. See Wu's paper [72] for an example of polynomial reduction. Both padding with zeros and polynomial modular reduction may be avoided using special cases of NTT-based convolutions. These approaches are known as wrapped convolutions.

With $F(X) = X^n - 1$, the product over R_{q_i} is performed without zero-padding and without polynomial reduction by computing:

$$\mathbf{c}_i = \mathrm{INTT}_{n,i}\left(\mathrm{NTT}_{n,i}(\mathbf{a}_i) \odot \mathrm{NTT}_{n,i}(\mathbf{b}_i)\right). \tag{2.5}$$

This convolution is known as Positive Wrapped Convolution (PWC).

When $F(X) = X^n + 1$, the product over R_{q_i} is also performed without zero-padding and without polynomial reduction. But this time, the input polynomials are weighted with a vector of the powers of a n-th primitive root of -1 over the finite-field $\mathbb{F}_i$. The output polynomial is also weighted but with a vector of the inverse powers of the n-th primitive root of -1. Let ψ_i be such a primitive root of -1, meaning that $\psi_i^n = q_i - 1 \mod q_i$ and for all $k < n$, $\psi_i^k \neq q_i - 1 \mod q_i$. The element ψ_i exists over $\mathbb{F}_i$ if and only if $2n = 1 \mod q_i$.

With $\Psi_i = (\psi_i^k)_{k=0}^{n-1}$ and $\Psi_i^{-1} = (\psi_i^{-k})_{k=0}^{n-1}$, the product over R_{q_i} is then performed by computing:

$$\mathbf{c}_i = \Psi_i^{-1} \odot \mathrm{INTT}_{n,i}\left(\mathrm{NTT}_{n,i}(\Psi_i \odot \mathbf{a}_i) \odot \mathrm{NTT}_{n,i}(\Psi_i \odot \mathbf{b}_i)\right). \tag{2.6}$$

This convolution is known as Negative Wrapped Convolution (NWC).

Comparison of NTT-based approaches. As previously mentioned, the efficiency of NTT-based polynomial ring multiplication stands upon the efficiency of Fast Fourier Transform adapted to NTT. Many different algorithms with asymptotic complexity of $O(n \log n)$ exist to perform such transforms. Each one of them has been proposed to address different size n and/or different computing architecture. When going for hardware implementation, one prefers to deal with transforms' size being a power of two. For simplicity, it is what we have considered in this thesis. Table 2.3 gives a high-level comparison of the different NTT-based approaches for multiplication over R_{q_i}.

The Positive Wrapped Convolution (PWC) approach seems to be the best from a computational and memory complexity point of view. The problem of PWC is the restriction of F to $X^n - 1$ that is not compatible with the FV scheme because it is not possible to find a cyclotomic polynomial of this form.

Comparing Padded Convolution (PC) and Negative Wrapped Convolution (NWC), the choice is in between: being able to have a finer choice over F while having to compute

Table 2.3: Comparison of NTT-based approaches to perform a polynomial multiplication over R_{q_i}. Considering n being a power of 2 and NTTs computed with the radix-2 Cooley-Tuckey algorithm. The table gives the numbers of multiplications and additions over the finite-field $\mathbb{F}_i$, the number of precomputed values, the restrictions over the choices of $F(X)$ and q_i, and if polynomial reduction is then required in our context. It is reminded that $N \geq 2n$.

Conv.	# of $\mathbb{F}_i$ mult	# of $\mathbb{F}_i$ add/sub	# of pre. values	Restrict. over $F(X)$	Restrict. over q_i	Poly. reduc.
PC	$3/2N \log N + 2N$	$3N \log N$	$N+1$	none	$N = 1 \mod q_i$	yes
PWC	$3/2n \log n + 2n$	$3n \log n$	$n+1$	$X^n - 1$	$n = 1 \mod q_i$	no
NWC	$3/2n \log n + 5n$	$3n \log n$	$2n+1$	$X^n + 1$	$2n = 1 \mod q_i$	no

polynomial reductions, or avoiding polynomial reductions while restricting F to $X^n + 1$. In favor of the first solution one should consider the batching method for binary plaintext being possible. In favor of the second, it is at this time the best performing approach to compute R_{q_i}'s products. In further discussions, the two approaches are considered as viable alternatives.

As previously expressed during their descriptions, these approaches bring some restrictions over the choice of the RNS basis elements. Hence, it appears necessary to study the feasibility of the RNS/NTT-coupled approach in our context.

2.3.3 Feasibility of the coupled approach

The restrictions on the feasibility of our approach mainly concern the RNS basis elements. Indeed, the Padded Convolution does not restrict the choice of $F(X)$ and therefore n. The other potential restriction on n is that $F(X)$ has to be a cyclotomic polynomial by definition of the FV scheme. This being inherent to the FV scheme, it is not dependent on the implementation approach.

Concerning the RNS basis elements, a first restriction is that they have to be mutually prime. A second restriction concerns their size, in order to limit the cost of the arithmetic in the RNS channel. Hence, their maximum size noted here s $(\max_{\mathcal{B}\cup\mathcal{B}'}\{\log_2 q_i)\}$ should be less than 64 bits. For convenience, all elements should also be roughly of the same size as long as this complies with the following restrictions.

A restriction inherent to the FV scheme is that the product of the element of the first basis $\mathcal{B}$ should have the appropriate size for security and correctness. Furthermore, the product of the element of the unified basis $\mathcal{B} \cup \mathcal{B}'$ should be larger than $2\delta q^2$. This is summarized by $\log_2 q = \sum_{\mathcal{B}} \log_2 q_i$ and $\log_2(2nq^2) < \sum_{\mathcal{B}\cup\mathcal{B}'} \log_2 q_i$, considering that δ is bounded by n.

The following restriction is brought by the NTT-based polynomial multiplication. Depending on the degree n of $F(X)$ and on the choice of PC or NWC, the restriction is different. In our case it is considered that the instantiated convolution's size is always a power of 2 for simpler hardware implementations. In the case of NWC, the degree n of handled polynomials is exactly a power of 2, and the restriction is $2n = 1 \mod q_i$. In the case of PC, the degree n of handled polynomials is not necessarily a power of 2 but the instantiated convolution is $(N > 2n)$. Hence, the restriction is in practice equivalent to NWC's one. Consequently, the

restriction on the RNS basis elements is: all element q_i in $\mathcal{B} \cup \mathcal{B}'$ should verify $2^m = 1 \mod q_i$, with 2^m equals to N for PC or $2n$ for NWC.

Finally, the last restriction is related to efficient modular arithmetic in the RNS channel. If the modular arithmetic is not relatively easy to perform, the RNS will not be beneficial.

All the identified constraints on RNS basis elements are summarized here:

constraint 1: they must be mutually prime.

constraint 2: they should be small ($\leq$ 64 bits) and roughly of same size for simpler hardware implementations.

constraint 3: there must be enough of them to verify $\log_2 q = \sum_{\mathcal{B}} \log_2 q_i$ and $\log_2(2nq^2) < \sum_{\mathcal{B} \cup \mathcal{B}'} \log_2 q_i$.

constraint 4: they must all verify $2n = 1 \mod q_i$ with n a power of 2.

constraint 5: it should be easy to perform modular arithmetic in the RNS channel they define.

Algorithm 2 Prime selection from NFLlib [95]

Input: s: prime size, m: margin bits, n: max polynomial degree, K: number of primes.
Output: $(p_0,\ldots,p_{K-1})$ a list of advantageous primes.

1: $\beta = 2^{s+m}$, $i = 1$, `primeList`=(), $t = 0$
2: **do**
3: $\quad c = \beta/2^m - i \cdot 2n + 1$
4: $\quad$ **if** `isPrime`(c) and $c > (1 + 1/2^{3m}) \cdot \beta/(2^m + 1)$ **then**
5: $\quad\quad$ append c to `primeList`
6: $\quad\quad t++$
7: $\quad$ **end if**
8: $\quad i = i + 1$
9: **while** $c > (1 + 1/2^{3m}) \cdot \beta/(2^m + 1)$ or $t < K$

Proposed selection of RNS basis elements. With respect to all the restrictions in the choice of RNS basis elements, the prime selection algorithm from the NFLlib [95] theoretically answers all our need. Algorithm 2 is a slightly modified version of the one proposed by Aguilar-Melchor et al. to fit with our notations and context.

The argument *margin bits* is here to force the m most significant bits of the selected primes to zero. This is here to allow lazy modular reduction while performing the NTT in software (see Harvey's work [96]). It could also be beneficial to reduce hardware cost, but this is not explored in this thesis.

Algorithm 2 allows to tune easily the size s of the desired primes, while ensuring that selected primes verify[I] $q_i = 1 \mod 2n$. Hence, constraints 1, 2 and 4 are verified.

The NFLlib is also providing a dedicated Barrett's modular reduction algorithm for the selected primes. This algorithm will be described in Chapter 3. Consequently, the last constraint that must be verified is their existence in sufficient number (and this for different prime's size s and different degree n).

We have implemented the Algorithm 2 using the GMP library for primality test. We then try to find as many primes as possible for s between 18 and 62, and polynomial degree n between 2^{10} and 2^{17}.

[I] Which is equivalent to verifying $2n = 1 \mod q_i$ when q_i is prime.

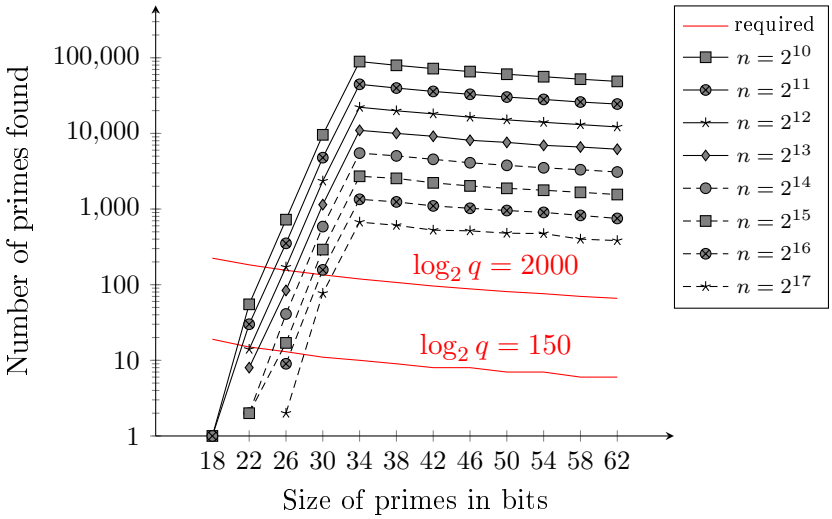


Figure 2.1: Number of primes of size s found with Algorithm 2 for different polynomial degree n. For NWC the degree of the polynomials F is exactly n, and for PC the degree of the polynomial is up to n. The continuous lines indicate the required number of prime to compose $\mathcal{B}$ and $\mathcal{B}'$ for different size of q.

Figure 2.1 shows the result of the experiment. For $s > 30$, the number of available primes to construct the RNS basis is clearly sufficient for all parameter sets studied in subsection 2.1.5. Another result we found in this experiment is that using smaller than 30-bit primes may result in difficulty to find enough primes for very large FV parameter sets. As we want an hardware approach that scale with as much FV parameter sets as possible, we will then consider primes of size at least 30 bits.

2.3.4 Concluding remarks on the RNS/NTT coupled approach

This section has presented the RNS/NTT coupled approach that we consider to address the performance bottleneck brought by polynomial ring multiplications. It has first contextualized the RNS representation and the NTT-based polynomial multiplication methods. Then the theoretical feasibility of the RNS/NTT coupled approach has been shown.

While describing the use of RNS for FV, concomitant works were mentioned to have adapted the primitives FV.Decrypt and FV.Mul&Relin. In addition to the feasibility of the full RNS variant of FV, we would like to consider the impact of these modifications on the computational performances of homomorphic evaluation. In particular, the profiling from [5] indicates that a non-negligible part of the performance complexity (20% to 38%) is located in RNS specific functions during FV.Mul&Relin.

The next section details the full RNS variant proposed by Halevi et al. [5], and discusses its profiling.

2.4 The full RNS variant of FV

As presented in subsection 2.3.1, the problem of the RNS in FV's context is the difficult adaptation of the scale-and-round operations involved in the decryption and multiplication

primitives. In 2016, Bajard et al. [4] proposed a first adaptation of these steps. The modifications involve a small increase in the noise growth that have an almost negligible impact on the parameter derivation. Their experimental results shows a speedup of at least 4 times for decryption and 1.7 times for multiplication compared to the schoolbook FV.

In 2018, Halevi et al. [5] proposed a simplified full RNS variant, with a reduced computational complexity compared to Bajard et al.'s one. Their variant is slightly more noisy, but still resulting in an almost negligible impact on parameters.

This section describes with our own words, the version implemented by Halevi et al. for completeness seek. A reader already familiar with this work can skip this section.

2.4.1 RNS base extension and RNS scale-and-round for FV

For efficient adaptation of the FV.Decrypt and FV.Mul primitives, one has to consider some RNS specific tricks. First, a *basis extension* that allows to perform the ciphertext tensor product in R during the FV.Mul primitive. Second, some methods that perform the *scale-and-round* operations inherent to FV.Decrypt and FV.Mul primitives.

Base extension. Let us consider an element $x \in \mathbb{Z}_q$ and its RNS representation in basis $\mathcal{B}$ noted $\{x_i\}_{i=1}^k$. The basis extension operation computes the residue of x for a new RNS basis element p_j from the initial knowledge of $\{x_i\}_{i=1}^k$.

Beforehand, note that the equality from Equation 2.1, expressing the reconstruction of x from $\{x_i\}_{i=1}^k$, may be rewritten as follow:

$$x = \Big(\sum_{i=1}^{k} \underbrace{[x_i\tilde{q}_i]_{q_i} \cdot q_i^*}_{\in[-\frac{q}{2},\frac{q}{2})}\Big) - v \cdot q, \text{ for some } v \in \mathbb{Z}_k = \left[-\frac{k}{2}, \frac{k}{2}\right). \tag{2.7}$$

We remind that $q_i^* = q/q_i \in \mathbb{Z}$ and $\tilde{q}_i = (q_i^*)^{-1} \in \mathbb{Z}_{q_i}$.

To compute a new residue $x'_j = [x]_{p_j}$ without going back to the positional representation of x, one has to compute the element v from Equation 2.7. The computation that gives v is:

$$v = \left\lfloor \sum_{i=1}^{k} \frac{[x_i\tilde{q}_i]_{q_i}}{q_i} \right\rceil. \tag{2.8}$$

This calculation is made in three steps in our case. First, compute for all i in $[1, k]$ the $y_i = [x_i\tilde{q}_i]_{q_i}$ (single precision modular arithmetic). Second, compute for all i in $[1, k]$ the $z_i = y_i/q_i$ (floating-point arithmetic). Third, accumulate all the z_i and round to the nearest integer to obtain v.

Once the element v is known, the new residue $x'_j = [x]_{p_j}$ is computed following the equation:

$$x'_j = [x]_{p_j} = \left[\sum_{i=1}^{k} y_i \cdot [q_i^*]_{p_j} - v \cdot [q]_{p_j}\right]_{p_j}. \tag{2.9}$$

This involves only single-precision modular arithmetic with only pre-computed values, beside the y_i's and v that depend on x. The pre-computed values are:

$$\big(\tilde{q}_i,\ [q_i^*]_{p_j}, \text{ and } [q]_{p_j}\big) \quad \forall\ (i,j) \in [1,k] \times [1,k']. \tag{2.10}$$

Repeat the computation of $[x]_{p_j}$ for all the p_j of a basis $\mathcal{B}'$ to perform the desired operation of basis extension.

The basis extension operation has been presented. Now, we consider the operations of scale-and-round for decryption and for multiplication. For this, a third expression of the equality of Equation 2.1 is required:

$$x = \Big(\sum_{i \in \mathcal{B}} \underbrace{x_i \cdot \tilde{q}_i \cdot q_i^*}_{\in [-\frac{q_i q}{4}, \frac{q_i q}{4})} \Big) - v' \cdot q, \text{ for some } v' \in \mathbb{Z}. \tag{2.11}$$

The scale-and-round operations involved in decryption and multiplication primitives are of the form: $y = \lfloor t/q \cdot x \rceil$. Depending on the case, the operand x and the required results are different. This results in two different situations that are treated in a similar way but involving different pre-computed values.

Scale-and-round operation for decryption. For the decryption primitive, the element x is in the interval $[-q/2, q/2)$ and the desired y should be returned modulo t.

Hence, by straightforwardly propagating the t/q factor in Equation 2.11, the RNS scale-and-round operation may be expressed as follow:

$$y = \left[\left\lfloor \frac{t}{q} \cdot x \right\rceil \right]_t = \left[\left\lfloor \sum_{i=1}^{k} x_i \cdot (\tilde{q}_i \cdot \frac{t}{q_i}) \right\rceil \right]_t. \tag{2.12}$$

Halevi et al. propose to pre-compute the $\tilde{q}_i t/q_i$ and decompose them into their integer and fractional parts:

$$\frac{\tilde{q}_i t}{q_i} = \omega_i + \theta_i, \text{ with } \omega_i \in \mathbb{Z}_t \text{ and } \theta_i \in \left[-\frac{1}{2}, \frac{1}{2} \right). \tag{2.13}$$

Consequently, the scale-and-round operation during decryption is simply performed by:

$$y = [w + v]_t, \text{ with } w = \left[\sum_{i=1}^{k} x_i \omega_i \right]_t \text{ and } v = \left\lfloor \sum_{i=1}^{k} x_i \theta_i \right\rceil. \tag{2.14}$$

Scale-and-round operation for multiplication. In the multiplication primitive, the element x is in $[-qp/2, qp/2)$ and the desired y should be returned modulo q. This requires the definition of extra terms related to $Q = qp$. For each $q_i \in \mathcal{B}$, $Q_i^* = Q/q_i = q_i^* p$ and $\tilde{Q}_i = [(Q_i^*)^{-1}]_{q_i}$. Similarly, for each $p_j \in \mathcal{B}'$, $Q_j'^* = Q/p_j = q p_j^*$ and $\tilde{Q'}_j = [(Q_j'^*)^{-1}]_{p_j}$. Hence, by straightforwardly propagating the t/q factor in Equation 2.11 we have the following:

$$\frac{t}{q} \cdot x = \left(\sum_{i=1}^{k} x_i \cdot \frac{\tilde{Q}_i p t}{q_i} + \sum_{j=1}^{k'} x'_j \cdot \tilde{Q}'_i p_j^* t \right) - v' p t. \tag{2.15}$$

Furthermore, this expression is nicely simplified considering it modulo each element p_j of the RNS basis $\mathcal{B}'$, namely:

$$\left[\left\lfloor \frac{t}{q} \cdot x \right\rceil \right]_{p_j} = \left[\left\lfloor \sum_{i=1}^{k} x_i \cdot \frac{\tilde{Q}_i p t}{q_i} \right\rceil + x'_j \cdot \left[\tilde{Q}'_j p_j^* t \right]_{p_j} \right]_{p_j}. \tag{2.16}$$

Hence, they propose to pre-compute in advance the $(\tilde{Q}_i p t/q_i)$'s and the $\left[\tilde{Q}'_j p_j^* t\right]_{p_j}$'s. Similarly than for decryption, the $(\tilde{Q}_i p t/q_i)$'s are decomposed them into their integer and fractional

parts $(\Omega_i + \Theta_i)$. The integer parts are reduced modulo each $p_j \in \mathcal{B}'$ and the fractional parts are directly stored as floating-points. Namely, all the precomputed values are:

$$\left(\Omega_{i,j} = [\Omega_i]_{p_j}\,,\ \Theta_i,\ \Lambda_j = \left[\tilde{Q}'_j p_j^* t\right]_{p_j}\right) \quad \forall\ (i,j) \in [1,k] \times [1,k']. \tag{2.17}$$

During multiplication, the RNS scale-and-round operation gives its results back in the basis $\mathcal{B}'$ (i.e. modulo p rather than modulo q). Namely, for each element $p_j \in \mathcal{B}'$:

$$y_j = [V + W_j]_{p_j},\ \text{with}\ V = \left\lfloor \sum_{i=1}^{k} x_i \Theta_i \right\rceil \ \text{and}\ W_j = \left[\Lambda_j x'_j + \sum_{i=1}^{k} x_i \Omega_{i,j}\right]_{p_j}. \tag{2.18}$$

To get back to $\mathbb{Z}_q$, a basis change is performed by executing the basis extension operation from $\mathcal{B}'$ to $\mathcal{B}$ followed by a deletion of the residue of the basis $\mathcal{B}'$. This final step requires the pre-computation of the following values:

$$\left(\tilde{p}_j,\ [p_j^*]_{q_i},\ \text{and}\ [p]_{q_i}\right) \quad \forall\ (i,j) \in [1,k] \times [1,k']. \tag{2.19}$$

All the RNS specific operations to adapt the FV primitives have been presented. The choice of Halevi et al. to use floating-point arithmetic at some point implies some possible approximation errors. In particular, to guarantee that their decryption procedure is correct considering these errors, they change the correctness requirement to: $\|\mathbf{v}\|_\infty < (\Delta - r_t(q))/4$. This is equivalent to considering one bit of additional noise in the ciphertext.

From an implementation point-of-view, it is important to note that, in the case of a polynomial, these operations must be performed for each coefficient. Hence, their is a high-level of parallelism accessible with respect to n. For large RNS basis, an additional level of parallelism may be achieve with respect to k and k'. But it is more limited than the previous one.

A last optimization of the full RNS variants of FV concerns a smart adaption of the relinearization primitive. This optimization, already introduced by Bajard et al., is briefly presented in next subsection.

2.4.2 Additional optimization

The optimization consists in modifying the relinearization key of FV to smoothly adapt to RNS the first version of the relinearization primitive. Namely, rather than using the decomposition basis T to mask the secret $\mathbf{s}^2$ (i.e. $T^i\mathbf{s}^2$), Halevi et al. propose to mask it with $\mathbf{s}_i^2 = \mathbf{s}^2\tilde{q}_i q_i^*$ (this implies that $[\mathbf{s}_i^2]_{q_j} = [\mathbf{s}^2]_{q_j}$ if $j = i$ and 0 otherwise).

Consequently, the new primitive is:

FV.RelinKeyGen($\mathtt{sk}$, $\mathcal{B}$):
for each $q_i \in \mathcal{B}$ sample $\mathbf{a}_i$ uniformly from R_q, $\mathbf{e}_i$ from χ.
Compute $\mathbf{r}_{0,i} = \left[-(\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i) + \mathbf{s}^2\tilde{q}_i q_i^*\right]_q$ and $\mathbf{r}_{1,i} = \mathbf{a}_i$.
Return $\mathtt{rlk} = \{(\mathbf{r}_{0,1}, \mathbf{r}_{0,2}, ..., \mathbf{r}_{0,k}), (\mathbf{r}_{1,1}, \mathbf{r}_{1,2}, ..., \mathbf{r}_{1,k})\}$.

After the multiplication primitive the polynomials of the non-canonical ciphertext ($\tilde{\mathtt{ct}} = (\tilde{\mathbf{c}}_0, \tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$) are in RNS representation according to the basis $\mathcal{B}$. Consequently, if we take $\tilde{\mathbf{c}}_{2,i} = [\tilde{\mathbf{c}}_2]_{q_i}$ the new relinearization is simply:

FV.Relin($\tilde{\mathtt{ct}}$, $\mathtt{rlk}$):
Compute $\hat{\mathbf{c}}_{2,0} = \left[\sum_{i=1}^{k} \mathbf{r}_{0,i}\tilde{\mathbf{c}}_{2,i}\right]_q$ and $\hat{\mathbf{c}}_{2,1} = \left[\sum_{i=1}^{k} \mathbf{r}_{1,i}\tilde{\mathbf{c}}_{2,i}\right]_q$.
Return $\mathtt{ct}_{mul} = \left([\tilde{\mathbf{c}}_0 + \hat{\mathbf{c}}_{2,0}]_q, [\tilde{\mathbf{c}}_1 + \hat{\mathbf{c}}_{2,1}]_q\right)$.

This concludes the adaptation of the FV scheme to the RNS representation. More information on the techniques and their consequences on noise growth can be found in the original papers. In the next subsection are presented the impact of these adaptations over the profiling of FV homomorphic evaluation complexity.

2.4.3 Profiling

In their paper, Halevi et al. present a profiling of their full RNS variant. In particular some timing for FV.Mul and FV.Relin and FV.Dec. The latter will not be presented because we are mainly interested in the FV primitives for homomorphic operations. In particular, they detail the profiling of the FV.Mul primitive that include their main contributions. The profiling expresses the computation workload with respect to basis extensions, scale-and-rounds and NTTs for Residue Polynomial Multiplications (RPMs).

Table 2.4: Profiling of the full RNS variant of the FV scheme implemented by Halevi et al. [5]. The FV.Mul&Relin primitive is detailed w.r.t. the time spent in the RNS specific functions of basis extension (Bext.) and scale-and-round (Sc&Rnd), and in the Residue Polynomial Multiplications (RPM).

							FV.Mult&Relin details		
L	n	S_q	k	Total ms	FV.Mul ms	FV.Relin ms	Bext. + Sc&Rnd	RPM	Others
1	2^{11}	55	1	3.57	3.16	0.41	37.2 %	61.2 %	1.6 %
5	2^{12}	110	2	12.7	10.1	2.58	30.3 %	67.8 %	2 %
10	2^{13}	220	4	57.6	38.9	18.7	25 %	72.4 %	2.6 %
20	2^{14}	440	8	252	174	78.3	28.3 %	69.5 %	2.2 %
30	2^{15}	605	11	887	555	332	26.3 %	71.1 %	2.6 %
50	2^{16}	1,026	19	4,434	2,368	2,066	26.7 %	70.5 %	2.8 %
100	2^{17}	2,042	38	29,884	12,890	16,994	21.6 %	75.2 %	3.2 %

In our case, we present the profiling from a slightly different angle. This is to highlight the part spent in RNS specific functions and the time spent to perform the equivalent of RPM. Hence, this requires some reasonable estimations to get the profiling presented in Table 2.4. Namely, it is estimated that 95% of the relinearization is spent performing the equivalent of RPM operations. In addition, it is considered that the internal products for RPM calculations represent 80% of the miscellaneous operations of their FV.Mul's detailed profiling.

This profiling shows that the main performance bottleneck of the FV.Mul&Relin primitive is the RPM operations which represent more than 60% of the computation workload. Nevertheless, compared to our first profiling in section 2.2, the relative time spent in the polynomial multiplications is reduced. Indeed, the additional operations to make the full RNS variant possible have also a non-negligible complexity.

Consequently, both RNS specific operations and Residue Polynomial Multiplications must be accelerated for an efficient implementation of the full RNS variants of the FV scheme.

2.5 Conclusion

In this chapter, we have detailed our analysis of the FV scheme towards its hardware acceleration. In a first time, we have presented its primitives and detailed the complexity of deriving FV parameters. In particular, the wide range of sizing parameters is inherently linked with the general use of the FV scheme.

In a second time, a profiling of a typical homomorphic evaluation has highlighted the computational complexity brought by large polynomial degree n and large modulus q. This is particularly the case during polynomial multiplications. Based on an analysis of the related works, we have chosen to explore the feasibility of the coupled approach RNS/NTT for the acceleration of FV.

The third section described more precisely the use of RNS representation and NTT-based polynomial multiplications in our context. It ends with a theoretical validation of the coupled approach according to all the prerequisites imposed by RNS and NTT.

The adaptation of the FV.Decrypt and the FV.Mul primitives to RNS brought by Bajard et al. and further simplified by Halevi et al. was described in the fourth section. The section concluded on the profiling showing the computational complexity partition of the full RNS FV.Mul&Relin primitive.

After this in-depth analysis, our acceleration strategy is defined by accelerating the two types of operations identified in the fourth section. Firstly, Residue Polynomial Multiplications (RPMs), and secondly, basis extension and scale-and-round operations. Due to computational complexity partition, our main focus is on RPMs.

As seen in Subsection 2.3.2, RPM operations may be performed through padded convolutions followed by polynomial reductions or through negative wrapped convolutions. In both case, the underlying NTTs require some precomputed values being dependent on the current finite-field (RNS channel). For convenience, we call a twiddle factor set the concatenation of the twiddle factors for a specific finite-field.

NTT operations are rather difficult to parallelize due to complex data access patterns making large NTT unfriendly for generic SIMD architectures. Hence, our choice is to explore dedicated hardware for accelerating these operations.

Our strategy is then to design basic blocks for the computation of NTT and the on-the-fly generation and usage of the twiddle factor sets. This is motivated by implementation issues highlighted by related works (more detailed in Chapter 3).

This strategy requires to address two main design issues. The first one is to be able to efficiently generate multi-field NTT circuits. Namely, NTT circuits that are able to perform transforms on different finite-fields without significant impact on performances. The second is to design a generator for the twiddle-factor sets of the different RNS channels without heavy consequences on hardware cost and on NTT circuit's throughput. In this thesis, we present a solution for each of the design issue. Chapter 3 presents our exploration of automatic generation of multi-field NTT circuits. Chapter 4 presents our generic architecture for the on-the-fly generation of twiddle factor sets.

Based on these contributions, the last chapter of this manuscript proposes a system level approach for the hardware acceleration of FV.

2.6 Annexes: details on FV primitives

We remind here the principal parameters of the FV schemes

- λ: security parameter. This parameter influences the choice of others with respect to theoretical and empirical hardness of the decision-RLWE problem.
- L: multiplicative depth. This parameter influences the choice of others with respect to correctness property.
- n: degree of the cyclotomic polynomials defining R.
- t: plaintext modulus defining R_t the plaintext ring.
- q: ciphertext modulus defining R_q the ciphertext ring.
- σ: error size of a normal distribution χ over R_q.
- T: decomposition base (relinearization version 1). For convenience, $l_T = \lfloor \log_T(q) \rfloor$.
- g: relinearization modulus (relinearization version 2).
- σ_g: error size of a normal distribution χ' over R (relinearization version 2).

Core primitives. The core primitives define the generation of the private and public keys, and the encryption and decryption processes.

- FV.SecretKeyGen(λ): sample $\mathbf{s}$ from R_2 with sufficient entropy with respect to security parameter λ, and output $\mathtt{sk} = \mathbf{s}$.
- FV.PublicKeyGen($\mathtt{sk}$): set $\mathbf{s} = sk$, sample $\mathbf{a}$ uniformly from R_q, $\mathbf{e}$ from χ and output $\mathtt{pk} = (\mathbf{p}_0, \mathbf{p}_1) = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})]_q, \mathbf{a})$.
- FV.RelinKeyGen:
  - V1 ($\mathtt{sk}$, T): set $\mathbf{s} = sk$.
    For $i \in [0; l_T]$ sample $\mathbf{a}_i$ uniformly from R_q, $\mathbf{e}_i$ from χ.
    Compute $\mathbf{r}_{0,i} = \left[-(\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i) + T^i \cdot \mathbf{s}^2\right]_q$ and $\mathbf{r}_{1,i} = \mathbf{a}_i$.
    Return $\mathtt{rlk1} = \{(\mathbf{r}_{0,0}, \mathbf{r}_{0,1}, ..., \mathbf{r}_{0,l_T}), (\mathbf{r}_{1,0}, \mathbf{r}_{1,1}, ..., \mathbf{r}_{1,l_T})\}$.
  - V2 ($\mathtt{sk}$, g): set $\mathbf{s} = sk$, and sample $\mathbf{a}$ uniformly from R_{gq}, $\mathbf{e}$ from χ'.
    Compute $\mathbf{r}_0 = [-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}) + g \cdot \mathbf{s}^2]_{gq}$ and $\mathbf{r}_1 = \mathbf{a}$.
    Return $\mathtt{rlk2} = (\mathbf{r}_0, \mathbf{r}_1)$.
- FV.Encrypt($\mathtt{pk}$, $\mathbf{m}$): for a plaintext element $\mathbf{m} \in R_t$, let $(\mathbf{p}_0, \mathbf{p}_1) = \mathtt{pk}$, and $\Delta = \lfloor \frac{q}{t} \rfloor$.
  Sample $\mathbf{u}$ uniformly from R_2, $\mathbf{e}_1$, $\mathbf{e}_2$ from χ.
  Compute $\mathbf{c}_0 = [\mathbf{p}_0 \cdot \mathbf{u} + \mathbf{e}_1 + \Delta \cdot \mathbf{m}]_q$ and $\mathbf{c}_1 = [\mathbf{p}_1 \cdot \mathbf{u} + \mathbf{e}_2]_q$.
  Return $\mathtt{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in R_q^2$.
- FV.Decrypt($\mathtt{sk}$, $\mathtt{ct}$): set $\mathbf{s} = \mathtt{sk}$, $(\mathbf{c}_0, \mathbf{c}_1) = \mathtt{ct}$ and return $\mathbf{m} = \left[\left\lfloor \frac{t}{q} \cdot [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \right\rceil\right]_t$.

Evaluation primitives. These primitives present basic operations for encrypted-computing.

- FV.Add($\mathtt{ct}_a$, $\mathtt{ct}_b$): set $(\mathbf{a}_0, \mathbf{a}_1) = \mathtt{ct}_a$ and $(\mathbf{b}_0, \mathbf{b}_1) = \mathtt{ct}_b$.
  Compute $\mathbf{c}_0 = [\mathbf{a}_0 + \mathbf{b}_0]_q$ and $\mathbf{c}_1 = [\mathbf{a}_1 + \mathbf{b}_1]_q$.
  Return $\mathtt{ct}_{add} = (\mathbf{c}_0, \mathbf{c}_1)$.

- FV.Mul($\mathtt{ct}_a$, $\mathtt{ct}_b$): set $(\mathbf{a}_0, \mathbf{a}_1) = \mathtt{ct}_a$ and $(\mathbf{b}_0, \mathbf{b}_1) = \mathtt{ct}_b$.
  Compute $\tilde{\mathbf{c}}_0 = \left[\left\lfloor \frac{t}{q}(\mathbf{a}_0\mathbf{b}_0)\right\rceil\right]_q$, $\tilde{\mathbf{c}}_1 = \left[\left\lfloor \frac{t}{q}(\mathbf{a}_0\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_0)\right\rceil\right]_q$ and $\tilde{\mathbf{c}}_2 = \left[\left\lfloor \frac{t}{q}(\mathbf{a}_1\mathbf{b}_1)\right\rceil\right]_q$.
  Return $\tilde{\mathtt{ct}} = (\tilde{\mathbf{c}}_0, \tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$.

- FV.Relin:

  - V1($\tilde{\mathtt{ct}}$, $\mathtt{rlk1}$): set $(\tilde{\mathbf{c}}_0, \tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2) = \tilde{\mathtt{ct}}$, and $\{(\mathbf{r}_{0,0}, ..., \mathbf{r}_{0,l_T}), (\mathbf{r}_{1,0}, ..., \mathbf{r}_{1,l_T})\} = \mathtt{rlk}$.
    Decompose $\tilde{\mathbf{c}}_2$ in base T: $(\tilde{\mathbf{c}}_{2,0}, ..., \tilde{\mathbf{c}}_{2,l_T})$.
    Compute $\hat{\mathbf{c}}_{2,0} = \left[\sum_{i=0}^{l_T} \mathbf{r}_{0,i}\tilde{\mathbf{c}}_{2,i}\right]_q$ and $\hat{\mathbf{c}}_{2,1} = \left[\sum_{i=0}^{l_T} \mathbf{r}_{1,i}\tilde{\mathbf{c}}_{2,i}\right]_q$.
    Return $\mathtt{ct}_{mul} = \left(\left[\tilde{\mathbf{c}}_0 + \hat{\mathbf{c}}_{2,0}\right]_q, \left[\tilde{\mathbf{c}}_1 + \hat{\mathbf{c}}_{2,1}\right]_q\right)$.
  - V2($\tilde{\mathtt{ct}}$, $\mathtt{rlk2}$): set $(\tilde{\mathbf{c}}_0, \tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2) = \tilde{\mathtt{ct}}$, and $(\mathbf{r}_0, \mathbf{r}_1) = \mathtt{rlk}$.
    Compute $\hat{\mathbf{c}}_{2,0} = \left[\left\lfloor \frac{1}{g}(\tilde{\mathbf{c}}_2\mathbf{r}_0)\right\rceil\right]_q$ and $\hat{\mathbf{c}}_{2,1} = \left[\left\lfloor \frac{1}{g}(\tilde{\mathbf{c}}_2\mathbf{r}_1)\right\rceil\right]_q$.
    Return $\mathtt{ct}_{mul} = \left(\left[\tilde{\mathbf{c}}_0 + \hat{\mathbf{c}}_{2,0}\right]_q, \left[\tilde{\mathbf{c}}_1 + \hat{\mathbf{c}}_{2,1}\right]_q\right)$.

- FV.Mul&Relin($\mathtt{ct}_a$, $\mathtt{ct}_b$, $\mathtt{rlk}$).
  Step 1: $\tilde{\mathtt{ct}}$ = FV.Mul($\mathtt{ct}_a$, $\mathtt{ct}_b$).
  Step 2: Return FV.Relin($\tilde{\mathtt{ct}}$, $\mathtt{rlk}$).

Chapter 3

Automatic generation of multi-field NTT architectures

This chapter presents a contribution for the definition of efficient basic blocks for Residue Polynomial Multiplications (RPMs) in the context of homomorphic cryptography. The problematic addressed here is the on-the-fly change of finite-field for a data-flow NTT circuit.

First, related works on hardware design for polynomial multiplication using the coupled approach RNS/NTT are discussed. Thus, our strategy will be motivated with respect to the implementation issues. Second, we highlight the capability of design space exploration of the SPIRAL DFT generator that is inherent to our strategy. Then our proposal of data-flow multi-field NTT is detailed, while showing the feasibility of the automatic generation of such designs. Finally, the hardware cost and the benefits of our proposal are presented.

3.1 Related works and strategy motivation

Before describing the related works, we remind that the pre-computed values for NTT and INTT are indifferently called twiddle factors. This twiddle factors are dependent on the finite-field over which the NTT is defined, and a twiddle factor set refers to the twiddle factors of a specific finite-field.

Now the terminology has been reminded, the related works on the implementation of RNS and NTT-based polynomial multiplications are presented.

In [79], Öztürk et al. proposed a RNS and NTT based polynomial multiplication. As their architecture is not pipelined, it cannot start a new residue polynomial multiplication before the previous one finishes. Its latency is then paid numerous time for the computation of a polynomial multiplication over R (as much as the size of the extended RNS basis). Furthermore, Öztürk et al. chose to pre-compute the different NTT twiddle factor sets on the host side, and send them along with the polynomial coefficients through the bus on which their accelerator is connected. Doing so, the communication cost between the host and the accelerator is doubled.

Cousins et al. [81] developed an Homomorphic Encryption Processing Unit to accelerate the LTV scheme, which is not scale-invariant like FV, but also has its bottleneck complexity in polynomial ring multiplications. They implemented a pipelined NTT as a primitive of the HEPU, and contrary to [79], they chose to store the NTT twiddle factors in ROM filled up at

compile time. As they point out, the storage capacity required for the different twiddle factor sets, one for each element p_i of the modulus chain, is quite important and uses a large part of the available BRAM on the targeted FPGA. This problem arises also for the FV scheme when polynomials are handled under RNS representation of their coefficients.

Sinha Roy et al. [78] present a co-processor (HE-processor) implementing building block operations for RLWE-based schemes, and in particular NTT and RNS primitives. They implement a memory access iterative NTT with improved routing of coefficients. They store in ROM only a subset of each required twiddle factor set and compute the others when needed. This results in a reduced memory requirement ($O(k \log_2(n))$) compared to [81] ($O(kn)$). Nevertheless, they note that the computation of the other twiddles inserts some bubbles into the NTT computation (up to $\sim 10,000$ bubbles for $n = 2^{16}$).

In view of implementation issues expressed in the literature, we choose to consider multi-field NTTs with on-the-fly computation of the twiddle factor sets. Thus, it requires the design of multi-field NTT circuits and of generators of twiddle factor sets. In this chapter we consider the former, and the latter is the subject of Chapter 4.

As already expressed, a NTT is similar to a Discrete Fourier Transform (DFT) in which complex arithmetic is replaced with modular arithmetic. With this in mind, our work explores the generalization of the hardware backend of the SPIRAL tool, from Milder et al. [97], to generate NTT designs in addition to DFT designs. Regarding time constraints, only data-flow architectures for NTT are considered in this thesis. Nevertheless, further work could explore the adaptation of our first solution to other type of DFT architecture generated by SPIRAL.

3.2 From SPIRAL DFT towards multi-field NTT designs

The SPIRAL project addresses the automation in software and hardware development for data signal processing. Thus, the DFT structure has already been explored in details forming an ideal starting point to generalize towards NTT implementations. For example, in his PhD thesis work, LingChuan Meng [98] explores the automatically generating tuned software libraries for modular polynomial multiplication. However, a similar extension to SPIRAL's hardware generation capability has not been explored; for example [97] focuses using SPIRAL to generate hardware for linear DSP transforms; while [99] generates hardware for sorting networks.

The perspective is to be able to express high level directives to an NTT design generator, allowing a system designer to tune the performances of its NTT according to application and system requirements. Tuned parameters could be related to lattice-based cryptosystem parameters, like NTT size n and manipulated word size s, or part of the implementation parameters like architecture type, radix size r or streaming width w.

During this thesis only fully-streaming DFT architecture has been studied and adapted to multifiled NTT architectures. Due to time constraints, we focused on the architecture type that has the highest throughput performances, as it is what is required by our application context.

3.2.1 Initial streaming DFT structure

This subsection presents the initial structure of streaming DFT transforms generated by the SPIRAL hardware back-end. In the same time, an overview of the SPIRAL tool is given. This is to consider the potential automation capability it brings in our approach. Most of descriptions and examples here are reformulations and contextualizations of Milder's papers [100, 97].

Matrix formulae. SPIRAL purpose is the automation of optimized software and hardware for data signal processing transforms. The starting point of an optimization is a dense[I] matrix representation of the considered transform. Namely, a transform of size n is represented by a matrix $n \times n$ (A_n) and the transformation of a n-sized vector $\mathbf{x}$ is simply the product $y = A_n \cdot \mathbf{x}$. For example, the discrete Fourier transform on n-point is defined as $\mathbf{y} = \mathrm{DFT}_n \cdot \mathbf{x}$. In this example, $\mathbf{x}$ and $\mathbf{y}$ are n-point complex vectors and $\mathrm{DFT}_n = \left[\omega_n^{kl}\right]_{0 \leq k,l < n}$, with $\omega_n = e^{-2i\pi/n}$.

Staying in a dense representation of the transform makes us compute $O(n^2)$ arithmetic operations to obtain the transformed vector $\mathbf{y}$. Hopefully, fast algorithms exist for many transforms, and the computation of y could be reduced to a quasi-linear arithmetic cost ($O(n \log n)$). The backbone behind SPIRAL optimization methods is the Kronecker's formalism that expresses fast algorithms as some particular decompositions of the dense matrices into a product of structured sparse matrices.

In SPIRAL, this decomposition is performed with the help of a formal language that represents algorithms with matrix formulae. Using the Backus-Naur formalism to describe the language, a matrix formula is simply defined as follow:

$$\begin{aligned}
\mathbf{matrix}_n &::= \mathbf{matrix}_n \cdots \mathbf{matrix}_n \\
&\mid \textstyle\prod_i \mathbf{matrix}_n \\
&\mid I_k \otimes \mathbf{matrix}_m && \text{, where } n = km \\
&\mid \mathbf{base}_n \\
\mathbf{base}_n &::= D_n = \mathrm{diag}(d_0, ..., d_{n-1}) \mid P_n \mid A_n
\end{aligned}$$

The first two lines state that a matrix formula can be decomposed into a product or an iterative product of matrix formulae. The second line $I_k \otimes \mathbf{matrix}_m$ expresses the decomposition of the initial matrix_n into k parallel instances of matrix_m ($n = km$). The matrix I_k is the matrix identity of size $k \times k$, and $\otimes$ is the Kronecker product. Finally, a matrix formula may simply refer to a generic terminal elements like a diagonal matrix D_n, a permutation matrix P_n, or some computational basic blocks matrix A_n.

For instance, the DFT transform of size 4 could be decomposed into DFTs of size 2 following the algorithm of Cooley-Tukey:

$$X = \mathrm{DFT}_4 \cdot x = (P_4 \cdot (I_2 \otimes \mathrm{DFT}_2) \cdot P_4 \cdot D_4 \cdot (I_2 \otimes \mathrm{DFT}_2) \cdot P_4) \cdot x$$

For a more detailed presentation of the formal language, the underlying permutations and/or example of concrete uses, please refer to [97].

The formal language is used by SPIRAL to rewrite a considered transform by selecting and combining some known fast algorithms under platform and/or user requirements. In the case of DFT, SPIRAL uses the Kronecker representations of Pease FFT, mixed-radix FFT

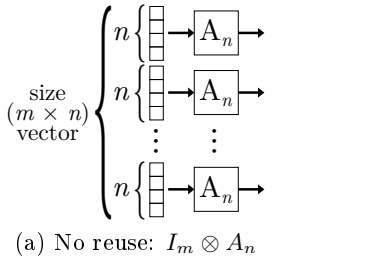
[I] Dense matrices are defined in opposition to sparse matrices which have most of their coefficients equal to zero.

and Bluestein FFT for its rewriting process. The resulting matrix formula then describes a specifically optimized fast algorithm for the considered transform.

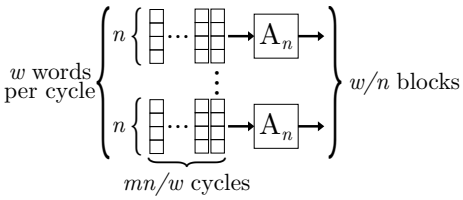
From an hardware implementation point of view, the matrix formula represents a theoretical data path computing the transform. To instantiate this data path, a correspondence between terminal elements and some hardware basic blocks is theoretically enough. Nevertheless, to simply make this correspondence mostly results in unfeasible data path for large sized problems. That is why the tool refines the matrix formulae with additional information.

Hardware formulae. The extended capability of SPIRAL brought by Milder et al. enriches the matrix formulae with hardware related information. This allows the propagation of hardware implementation decisions into the matrix formulae. An enriched formula is called a hardware formula.

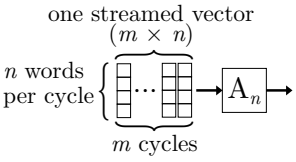
Basically, this enrichment simply adds the information of sequential reuse of basic RTL blocks. Milder et al. defines two types of sequential reuse: streaming and iterative reuses. The streaming reuse enriches Kronecker product formulae in order to add the information of



(a) No reuse: $I_m \otimes A_n$

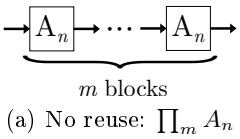


(b) Partial reuse: $I_{mn/w} \otimes^{sr} \left(I_{w/n} \otimes A_n\right)$

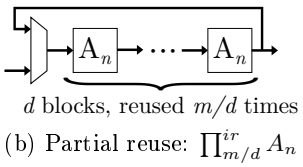


(c) Full reuse: $I_m \otimes^{sr} A_n$

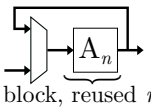
Figure 3.1: Example of streaming reuses.



(a) No reuse: $\prod_m A_n$



(b) Partial reuse: $\prod^{ir}_{m/d} A_n$



(c) Full reuse: $\prod^{ir}_{m} A_n$

Figure 3.2: Example of iterative reuses.

reuse in time of some basic blocks, rather than full parallelism in space. Figure 3.1 presents this streaming reuse principle. A Kronecker product may be enriched with full, partial or no streaming reuse decisions.

The iterative reuse enriches an iterative product on identical formulae in order to express reuse of the same data path multiple times. This is implemented in hardware with feedback mechanisms. As seen in Figure 3.2 an iterative product formula may be enriched with full, partial or no iterative reuse decisions.

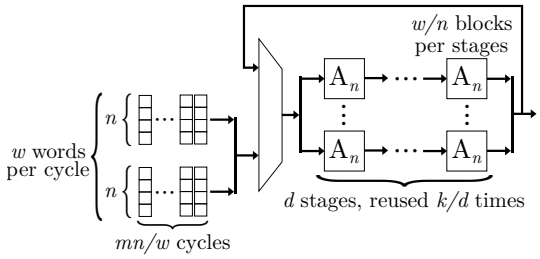


Figure 3.3: Combined streaming and iterative reuse: $\prod_{k/d}^{ir}\left(\prod_{d}\left(I_{nm/w}\otimes^{sr}\left(I_{w/n}\otimes A_n\right)\right)\right)$

Finally, iterative and streaming reuse may be combined to enrich more complex matrix formulae. An example of mixed sequential reuse is given in Figure 3.3.

These design decisions have consequences in terms of resource utilization, throughput and latency, which are propagated from terminal basic blocks up to the overall transform formula. As a consequence, the SPIRAL tool is able to explore multiple implementation solutions, and choose one meeting some system level requirements.

The presentation of the formula rewriting processes highlights the design space exploration capability of the SPIRAL hardware backend. Its use to generate NTT designs is seen as a guarantee of flexibility in our application context. In particular, it would be particularly interesting to be able to generate NTTs of different sizes without significant additional development costs.

Our discussion is now mainly focused on the DFT basic blocks used by SPIRAL. The purpose is to define the required basic blocks for the generation of multi-field NTT.

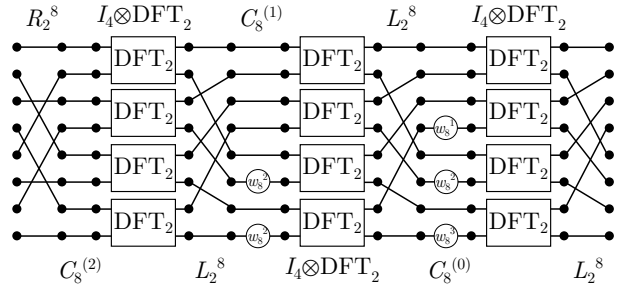
DFT basic blocks. The SPIRAL DFT hardware generation starts its formula rewriting processes from the Pease FFT algorithm.

$$\mathrm{DFT}_{r^t} = \left(\prod_{l=0}^{t-1} L_r^{r^t}(I_{r^{t-1}}\otimes \mathrm{DFT}_r)C_{r^t}^{(l)}\right)P_r^{r^t} \tag{3.1}$$

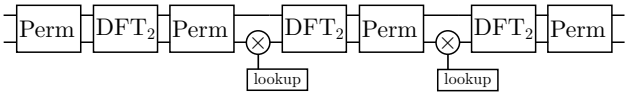
The Kronecker formulation of the decimation in frequency Pease FFT algorithm is presented in equation 3.1. The algorithm parameter r is the radix basis of the transform. This parameter has an influence on the possible size of the DFT (it should be a power of r, namely $n = r^t$). The algorithm is then decomposed in t number of stages ($t = \log_r(n)$), performed after an initial permutation $P_r^{r^t}$ (the operation's order is from right to left).

The matrices $C_{r^t}^{(l)}$ are diagonal matrices that represent multiplications with the twiddle factors. Exponent l expresses that these matrices are dependent of the considered stages $l \in 0, ..., t-1$, in contrary to the matrices $L_r^{r^t}$ that are only dependent on r and t. The matrix DFT_r is a basic bloc implementing an r-point DFT. The matrices $L_r^{r^t}$ are permutation matrices that prepare the stage's outputs for the next stage.

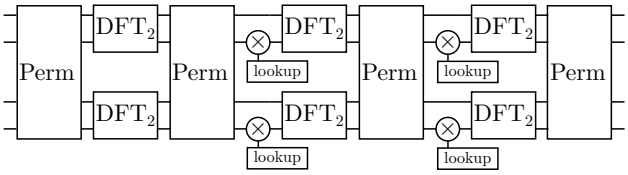
Both streaming and iterative reuse can be applied on this matrix formula. The kronecker product $I_{r^{t-1}}\otimes\mathrm{DFT}_r$ could be streamingly reused with any width w such that w is a multiple of r and divides r^{t-1}. The iterative $\prod_{l=0}^{t-1}(...)$ could be iteratively reused with depth d that divides t. Figure 3.4 presents some illustration of different reconstruction of equation 3.1 to match sequential reuse directives. Contrary to the matrix formula, the schematics are read from left to right.



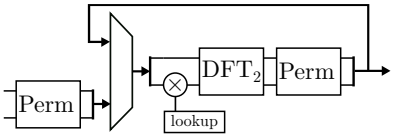
(a) No sequential reuse



(b) Streaming reuse, width $w = 2$



(c) Streaming reuse, width $w = 4$



(d) Mixed reuse, streaming width $w = 2$ and depth $d = 1$

Figure 3.4: Illustration of sequential reuse impact on a DFT circuit. Example for a radix-2 DFT on 8 points: $\mathrm{DFT}_{2^3} = \left(\prod_{l=0}^{2} L_2^8 (I_4 \otimes \mathrm{DFT}_2) C_8^{(l)}\right) P_2^8$.

For the resulting DFT hardware formulae to be transformed into proper HDL description, the SPIRAL hardware backend should be able to generate some generic basic blocks. For the generation of streaming permutations appeared after streaming reuse rewriting, Püschel et al. [101] proposed a solution that is used in SPIRAL.

Regarding the diagonal matrices $C_{rt}^{(l)}$ the generation is simply done by instantiating a multiplier on each streaming way that requires multiplication with twiddle factors different from one. Due to streaming reuse rewriting, some instantiated multipliers may have to access different twiddle factors. Hence, a lookup table storing all required twiddle factors is instantiated along with the multiplier.

Finally, the DFT_r basic blocks are just the decimation in frequency version of the radix-r butterflies.

From the previous presentation, two observations are important for our objective. First, besides the basic matrix DFT_r and the diagonal matrices $C_{rt}^{(l)}$, nothing is under the influence of arithmetic and only reflects the structure of the algorithm. Second, SPIRAL has full knowledge of which twiddle factors are involved in each stage of the DFT circuit.

Hence, to generate NTT rather than DFT, we simply have to replace the complex arithmetic by modular arithmetic, and replace complex twiddle factors in lookup table by NTT twiddle factors. No structural change of the algorithm is required. In the next subsection, the modular arithmetic that we choose to implement is presented.

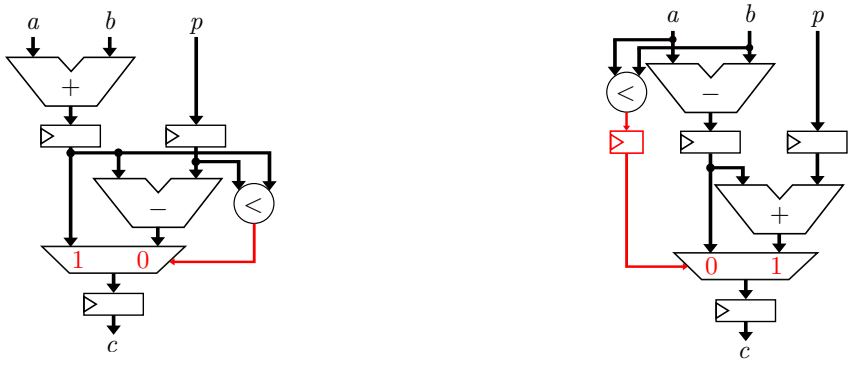
3.2.2 Finite-field arithmetic

The previous subsection has presented the DFT generation capability of the SPIRAL hardware backend. It was pointed out that it is sufficient to replace complex arithmetic by modular arithmetic and complex twiddle factors by NTT twiddle factors to generate NTTs in place of DFTs.

In this section we present the modular arithmetic operators required to perform an NTT over a finite-field. The finite-fields considered are the $\mathbb{Z}_{p_i} = \mathbb{Z}/p_i\mathbb{Z}$, with p_i some s-bit primes over which the n-point NTT is defined.

The modular arithmetic is dependent of the modulus p_i considered. We remind that the modulus p_i are the RNS basis element in our context. Consequently, they are chosen using the NFLlib prime selection algorithm presented in Chapter 2.

Modular additions and subtractions. The modular additions and modular subtraction are defined by: $c_{add} = a + b \mod p_i$ and $c_{sub} = a - b \mod p_i$, with $a, b \in \mathbb{Z}_{p_i}$. As the input operands a and b are elements of the finite-field, it implies $0 \leq a, b < p_i$. Hence the modular addition and subtraction are simply performed with one comparison and at most two additions/subtractions.



(a) Modular addition

(b) Modular subtraction

Figure 3.5: Modular additions and subtractions over $\mathbb{Z}_{p_i}$

Figure 3.5 gives some examples of hardware implementation for modular adder and modular subtracter. In order to develop generic hardware basic blocks, the modulus p_i is also considered as an input operand. The proposed operators are pipelined, thus have two cycles latency and output one result every cycle.

Modular multiplications. A multiplication over $\mathbb{Z}_{p_i}$ is defined by $c_{mul} = a * b \mod p_i$. Even if we know the dynamic range of the input operand, the modular reduction is not necessarily trivially performed after a multiplication. As discussed in Chapter 2, the modulus p_i is selected using the prime selection algorithm from NFLlib [95] (see Algorithm 2). This library

gives also a Barrett-like reduction algorithm that takes advantage of the prime characteristics. Consequently, this reduction has been chosen to implement a basic block for modular multiplication.

Algorithm 3 Modular multiplication from NFLlib [95]

Input: $a, b \in [0\ ,\ p_i)$, p_i selected with Algorithm 2, $v_i = \lfloor \beta^2/p_i \rfloor \mod \beta$ $(\beta = 2^{s+m})$
Output: $r = a \times b \mod p_i$
1: $q \leftarrow v_i \cdot u_1 + 2^m \cdot u \mod \beta^2$
2: $r \leftarrow u - \lfloor q/\beta \rfloor \cdot p_i \mod \beta$
3: **if** $r \geq p_i$ **then**
4: $\quad r \leftarrow r - p$
5: **end ifreturn** r

The modified Barrett reduction from NFLlib is presented in Algorithm 3. It requires a $(s+m)$-bit reciprocal related to the modulus p_i ($v_i = \lfloor 2^{2(s+m)}/p_i \rfloor \mod 2^{(s+m)}$. Both the prime p_i and the reciprocal v_i are considered as input operands in the definition of a generic modular multiplier basic block.

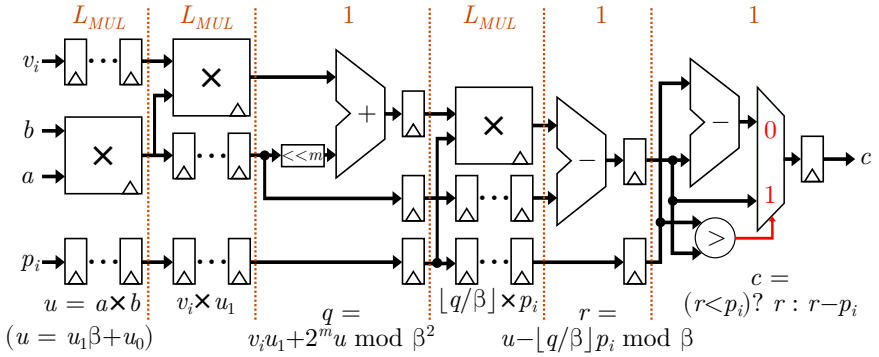


Figure 3.6: Hardware schematic of the NFLlib modular multiplication algorithm

Figure 3.6 presents the schematic of the proposed basic bloc modular multiplier. The operator is fully pipelined and has a latency $L_{MMUL} = 3L_{MUL} + 3$ with L_{MUL} the latency of a s-bit multiplier. Signal widths have not been noted on the schematic, but only the appropriate widths are used at each operations. For example the third multiplication only requires the $s+m$ lower bits of the adder results. The total hardware cost of this operator is dominated by the three multipliers and the pipeline registers.

Due to time constraint, we have implemented the modular multiplication rather straightforwardly. Some additional works could explore the optimization of this design to reduce its latency and its hardware cost. Any improvement on the modular multiplication block could be beneficial for the overall NTT design.

Now that we have presented the modular arithmetic basic blocks, the last requirement for multi-field NTT generation is to handle multiple NTT twiddle factor sets. A solution to this design problematic is presented in next subsection.

3.2.3 Modification of twiddle factors handling

For DFT hardware generation, SPIRAL pre-computes the twiddle factors and stores them in the different lookup tables (ROM) at compile time. This approach is suitable for DFT as the twiddle factors do not change over time. Similarly, if one has only a single finite-field on which performing the NTT, this solution is the most suitable. But in our case, the RNS representation imposes us to find a way to reprogram these lookup tables without major impact on performances.

Reminder on twiddle factor sets. In the case of DFT, the twiddle factors are the first $n/2$ powers of the n-th primitive root of unity over $\mathbb{C}$ (i.e. $\left(e^{(2k\pi/n)}\right)_{0\leq k<n/2}$). Similarly for NTT, the twiddle factors are the powers of a n-th primitive root over the finite-field $\mathbb{Z}_{p_i}$. As a reminder, the definition of a n-th primitive root over $\mathbb{Z}_{p_i}$ is an element ω_i such that $\omega_i^n = 1 \mod p_i$ and $\omega_i^k \neq 1 \mod p_i$ for $1 \leq k < n$. Hence the twiddle factors required for an NTT over the finite field $\mathbb{Z}_{p_i}$ are $\Omega_i = \left\{\omega_i^k \mod p_i\right\}_{0\leq k<n/2}$.

For inverse DFT, the twiddle factors are the multiplicative inverse of those for forward DFT (i.e. $\left(e^{-(2k\pi/n)}\right)_{0\leq k<n/2}$). Similarly, the twiddle factors for inverse NTT are $\Omega_i^{-1} = \left\{\omega_i^{-k} \mod p_i\right\}_{0\leq k<n/2}$.

Finally, the n-point inverse DFT requires the precomputed multiplicative inverse of n over the complexes in order to scale down the outputted vector. The inverse NTT also requires this multiplicative inverse of n over the considered finite-field noted $n_i^{-1} = n^{-1} \mod p_i$. For simplicity, this is implicit in the following discussions.

Hence, beside the final scaling, the forward NTTs and the inverse NTT share the same architecture. The only difference is in the twiddle sets, namely Ω_i for the forward one and Ω_i^{-1} for the inverse one.

The distribution of the twiddle factor sets in the different lookup tables is imposed by the structure of the Pease FFT algorithm. Each stage of the resulting NTT circuit may embed up to $w/2$ multipliers and memory elements to store the required twiddles for this stage.

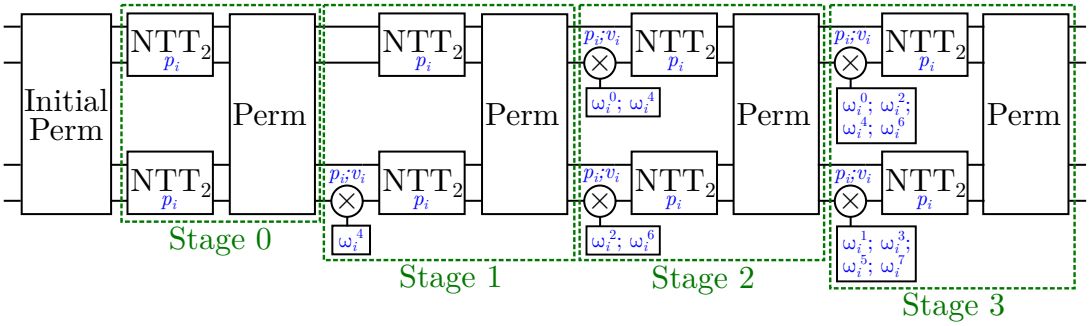


Figure 3.7: Example of the initial twiddle factors partition for a fully-streaming NTT_{2^4} with streaming width equal to 4. The finite-field specific values are highlighted in blue.

As an illustration, Figure 3.7 presents the composition of the different lookup table for a fully-streaming 16-point NTT with a streaming width of 4 elements per cycle. The other finite-filed specific values are colored in blue. Namely, the prime p_i and the reciprocal v_i that are required for the arithmetic operations.

The distribution of the twiddle factors accross the NTT circuit is known by the SPIRAL hardware generator. Hence, rather than implementing the memory elements directly in the data path, our proposed solution disassociates the twiddle memory elements of their stages, making them programmable, and handling them as a bank of memory elements. From now on, a twiddle bank refers to the concatenation of all twiddle memory elements for a specific RNS channel. In addition to the twiddle factors, a twiddle bank will also store the other finite-field specific values, namely the prime p_i and its reciprocal v_i. Each twiddle bank stores a twiddle factor set and a (p_i, v_i) pair of one RNS channel at a time, and could be reprogrammed with a new set when required.

To not insert bubbles during the reprogramming of a twiddle bank, our solution implements a circular access and reprogramming of G different twiddle banks. This number G is related to the maximum number of simultaneous RNS channels in the NTT data path. If we note $T = n/w$ the throughput of the NTT data path and lat_{NTT} its latency, hence the architecture instantiates $G = \lceil lat_{NTT}/T \rceil + 1$ of them.

The main modification for the generated NTT data path is that it has to consider all the finite-field specific values as inputs. The resulting multi-field design is presented in the next section.

3.3 Proposition of a multi-field NTT design

The section presents our proposed multi-field NTT architecture by means of cyclical access and reprogramming of twiddle banks. As presented in section 5.3 the generation of the different twiddle factor sets is made outside of the NTT circuit.

A global design overview is given in a first subsection, introducing in particular the distinction between the twiddle path and the data path. The data path characteristics are detailed in a second subsection, and the twiddle path's ones in a third subsection.

3.3.1 Design overview

The proposed solution makes the distinction between the data path and the twiddle path. The data path implements the NTT algorithm itself, and the twiddle path handles the different twiddle factor sets. This is illustrated in Figure 3.8.

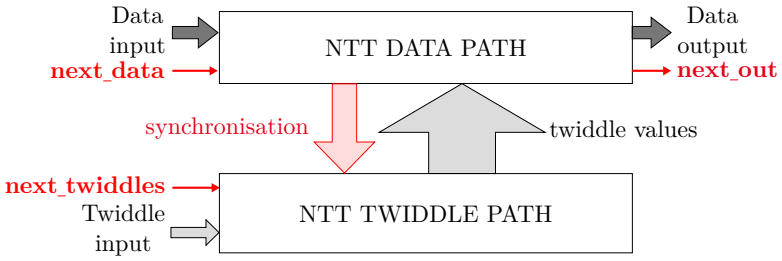


Figure 3.8: Number Theoretical Transform (NTT) flow. Schematic for $w = 2$.

The two paths have their own distinct flows. The twiddle path receives new twiddle factor sets one cycle after it has been announced by the **next_twiddles** signal. The data path begins a new NTT computation one cycle after the reception of the **next_data** signal The end of the computation is signaled by the **next_out** signal.

To achieve a proper functioning of the overall design, a twiddle factor sets must be given to the twiddle path some time before the corresponding NTT computation start. This is to take into account the programming of the twiddle factor sets into a twiddle bank. The data path is then responsible of sending synchronization signals towards the twiddle paths. The twiddle path consequently feeds the data path with the appropriate twiddle factors.

The data path is generated by SPIRAL to which has been added the basic NTT blocks by specifying the modular arithmetic operators presented in previous section. For a proof of concept solution of the multi-field NTT generation, the twiddle path is generated by a python script.

The next subsections present in details the SPIRAL generated data path and its characteristics, and how the twiddle path is generated.

3.3.2 Data path

The data path simply performs the permutations and the arithmetic operations to compute the NTT. The computation is performed in a single flow from the first stage to the last stage.

Each stage of the NTT data path is responsible of sending synchronization signals to the twiddle path when it starts its computation over a different finite-field. Consequently, the twiddle path connects the concerned stage with the proper twiddle bank storing the required twiddles and pre-computed values.

multi-field NTT stage. A NTT stage consider all the finite-field specific values as inputs. Each step has different requirements in terms of expected finite-field values, but the principle of access to them is always the same.

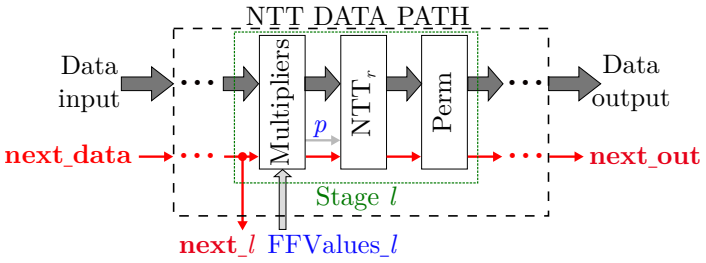


Figure 3.9: Stage structure for a NTT stage.

Figure 3.9 presents the basics of a NTT stage in our solution. The synchronization signal **next_**l is extracted from the control flow directly coming from the **next_data** signal. Hence, when **next_**l is asserted it tells the twiddle path that the stage l will have data of a new finite-field incoming on the next clock cycle. Upon this information the appropriate finite field values (**FFValues_**l) are fed to the stage.

For the proper generation of the twiddle path, the finite-field values expected by each stage of the NTT must be be known. Hence, when SPIRAL generates the NTT data path, some information on its resulting structure should be propagated to our twiddle path generator.

Information for twiddle path generator. For automatic generation of the twiddle path, the SPIRAL tool should precise the structure of the NTT data path. Namely, it consists in the number of stages of the NTT, and for each one, the required finite-field specific elements. In these required elements are included the prime p_i, the reciprocal v_i (if needed), the different

memory elements for the required twiddle for this stage (if necessary), and for each memory element of a stage, the powers of the required twiddle factors.

Each stage is identified by an index $l \in [0, t-1]$, with $t = \log_r n$. Each memory element of a stage is identified by an index $m \in [0, w/2-1]$, which indicates on which pair of streaming way is located the associated multiplier on the data path.

For instance, here is the information expected by the twiddle path generator for the circuit in Figure 3.7:

- stage $l = 0 \rightarrow$ (p_i, no reciprocal, no memory element).
- stage $l = 1 \rightarrow$ (p_i, v_i, 1 memory element):
— memory element $m = 1 \rightarrow$ twiddle powers $[4]$
- stage $l = 2 \rightarrow$ (p_i, v_i, 2 memory elements):
— memory element $m = 0 \rightarrow$ twiddle powers $[0, 4]$
— memory element $m = 1 \rightarrow$ twiddle powers $[2, 6]$
- stage $l = 3 \rightarrow$ (p_i, v_i, 2 memory elements):
— memory element $m = 0 \rightarrow$ twiddle powers $[0, 2, 4, 6]$
— memory element $m = 1 \rightarrow$ twiddle powers $[1, 3, 5, 7]$

Note that a memory element is uniquely identified in the twiddle bank by the pair of indexes (l,m). With this information, the interface between the data path and the twiddle path is straightforward, as well as for the structure of a twiddle bank. For this example, a twiddle bank is composed of:

- 1 register for the prime p_i.
- 1 register for the reciprocal v_i.
- 1 register indexed ($l = 1$, $m = 1$).
- 2 memory of depth 2: one indexed ($l = 2$, $m = 0$) and the other ($l = 2$, $m = 1$).
- 2 memory of depth 4: one indexed ($l = 3$, $m = 0$) and the other ($l = 3$, $m = 1$).

And the **FFValues** signals are:

- **FFValues**_0 = {**p**_0}
- **FFValues**_1 = {**p**_1, **v**_1, **tw**_1_1}
- **FFValues**_2 = {**p**_2, **v**_2, **tw**_2_0, **tw**_2_1}
- **FFValues**_3 = {**p**_3, **v**_3, **tw**_3_0, **tw**_3_1}

The last remaining information required by the twiddle path generator is how the read-accesses for the twiddles in memories are synchronized with the NTT data path. Actually, the order in which the twiddle factors are read from the memories is the same as in the original DFT design. Hence, the address generators already generated by SPIRAL are reused in our solution. More details are given when presenting the twiddle path.

In this subsection, we have presented the principle of the fully-streaming multifiled NTT data paths. In particular, we have described how each stage of the NTT synchronizes itself with the twiddle path. In addition, we have highlighted the information required by the twiddle path generator to generate a twiddle path consistent with the data path. Its internal structure and the functioning of the twiddle path is presented in next subsection.

3.3.3 Twiddle path

The twiddle path handles different twiddle factor sets from up to a maximum of G different finite-field. Each of the G twiddle factor sets is stored in a different twiddle bank. It is reminded that a twiddle bank is the concatenation of memories and registers required by the data-path to store a single twiddle factor set.

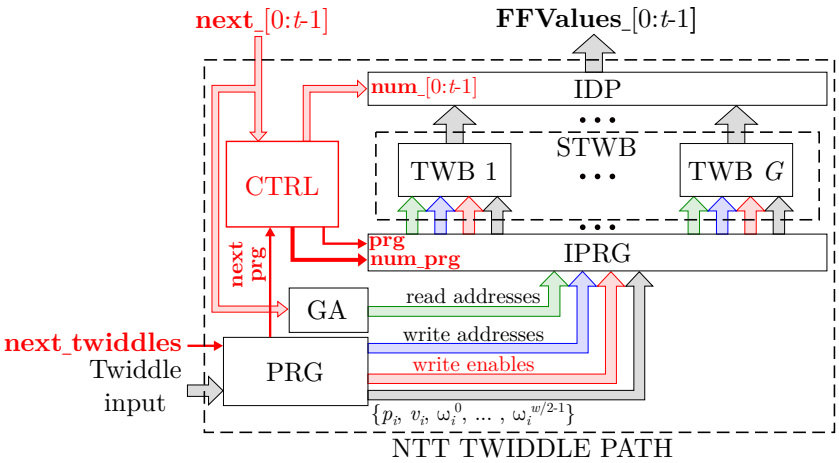


Figure 3.10: Twiddle path internal structure.

The principle of the twiddle path is described here. The description refers to Figure 3.10. The twiddle path cyclically reprograms the different twiddle banks whenever a new twiddle factor set is inputted (signaled by **next_twiddles**). It cyclically feeds the different stages with their expected finite-field values (**FFValues_**l) when requested by the corresponding **next_**l signals. To guarantee a collision-free access and reprogramming of the twiddle banks, the number of instantiated twiddle banks G is one plus the maximum number of RNS channels simultaneously present in the data path.

The G different twiddle banks (TWB [1:G]) are instantiated in the STWB module. On one side, this set of twiddle banks is connected with a data path interconnect (IDP). On the other side, it is connected with a program interconnect (IPRG).

The IDP module is controlled by the CTRL module under the influence of the synchronization signals coming from the data path. Similarly, the IPRG module is controlled by the CTRL module under the influence of the **next_prg** signal issued by the programming module (PRG).

The PRG module is responsible of the reprogramming of the twiddle banks. This programming is requested at each input of a new twiddle factor set, signaled by the **next_twiddles** signal. The PRG module signals to CTRL that a new program flow is coming with the **next_prg** signal.

The twiddle banks not currently programmed are accessed accordingly to the read address signals generated by the GA module. For each memory in a twiddle bank, the generation of read address signals is reset by the control signals from the data path flow.

Programming a twiddle bank. We remind that a twiddle bank (TWB) is the concatenation of all the twiddle memories and registers required by the data path, plus a pair of registers storing the (p_i, v_i) pair.

The programming of a twiddle bank is performed with the same throughput as the data path throughput. We remind that the throughput of a fully-streaming NTT in our case is $T = n/w$. Namely a new n-point NTT is performed at least every T cycles. Moreover, it is reminded that their is exactly $n/2$ different twiddle factors for a n-point NTT. Consequently, it is sufficient for the programming phase to have a streaming flow of $w/2$ twiddles per cycle.

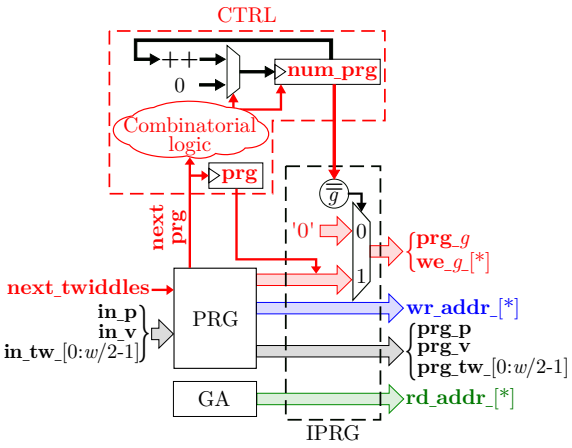


Figure 3.11: Principle of the program interconnect module (IPRG).

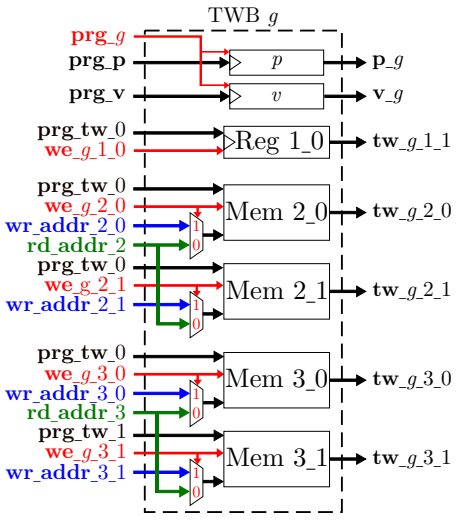


Figure 3.12: Twiddle bank internal structure. Example for $n = 16$ and $w = 4$.

To reprogram a twiddle bank, the pair (p_i, v_i) and the twiddle factors $\{\omega_i^j\}_{0 \leq j < n/2}$ are sent through PRG along with appropriate **write address** and **write enable** signals for each storing location of the bank. As seen in Figure 3.11, the bank currently programmed receives the **we_g_**[*] signals from PRG: bank number g is reprogrammed when **num_prg** is equal to g. Other banks are only addressed for reads, using the simple mechanism of address selection in figure 3.12.

The choice of the bank currently programmed is done by cyclically updating the **num_prg** register in $\{1, ..., G\}$ with the arrival of a new programming phase, signaled with **next_prg** going high for one cycle.

Generating the program flow. The twiddle path generator knows, for each memory element of a twiddle bank, which twiddle factors are expected. In addition to this, the twiddle path generator has to know how the twiddle factor sets are inputted to the PRG module. This depends on how the twiddle generator presented in Chapter 4 outputs the values.

in_tw_0	⟶	0	⋯	k*$w/2$	⋯	(T-1)*$w/2$
in_tw_1	⟶	1	⋯	k*$w/2$+1	⋯	(T-1)*$w/2$+1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
in_tw_($w/2$-1)	⟶	$w/2$-1	⋯	k*$w/2$+$w/2$-1	⋯	(T-1)*$w/2$+$w/2$-1

Figure 3.13: Input formalism for the different twiddle factor sets.

Actually, the twiddle factor sets are inputted in increasing order of powers, namely $\{\omega_i^0, \omega_i^1, ..., \omega_i^{n/2}\}$. Moreover, they are inputted $w/2$ elements per cycles in order to respect the

required programming throughput. Figure 3.13 illustrates this formalism which is respected by the twiddle generator presented in Chapter 4.

From now on we call a bunch of twiddle the $w/2$ consecutive twiddles received in a cycle. Hence, to uniquely identify a specific twiddle factor in the input flow, all that is required is: its bunch number $k \in [0 : T-1]$ and its index in the bunch $i \in [0 : w/2-1]$. Consequently the PRG module should know, for each memory element of a twiddle bank, the (k,i) pairs of the expected twiddles.

$$\text{tw}(k,i) = \omega_i^{\frac{kw}{2}+i}, \quad \forall (k,i) \in [0, T-1] \times [0, w/2-1]. \tag{3.2}$$

In our case[I], due to the well structured distribution of the twiddles into the different registers and memories of a twiddle factor bank, each memory element expects its twiddles from a single bunch index i. Namely, the bunch index from which the twiddles are extracted is given by $i = \text{power} \mod w/2$, with "power" being the power of any twiddle for that memory element.

Similarly, each memory element expects its twiddles from the bunches of index k following a friendly arithmetic progression with the common difference being a powers of two.

Consequently, for each memory element of a twiddle bank, the desired PRG module's behavior is easily implemented using a counter with an initial offset value and a step value. These counter's characteristics can be expressed in function of the stage index l and the streaming way index m that identify the memory element.

$$\text{offset}(l,m) = \frac{m \cdot n}{2^{(l+1)} \cdot w/2}$$

$$\text{step}(l) = \begin{cases} 0 & \text{if the storage element is a register.} \\ \frac{w \cdot n}{2^{(l+2)} \cdot w/2} & \text{if the storage element is a memory.} \end{cases}$$

Note that only the counters with step values larger than one are actually implemented. Indeed, the offset values represent the first bunches from which twiddles are extracted. A step value of 0 indicates that only one twiddle is extracted from the flow as it feeds a register in the twiddle bank. A step value of 1 represents that a twiddle is extracted from the flow at each cycle. Hence, all the information needed to extract the concerned twiddles may be centralized in a single global counter. Furthermore, a global counter is implemented anyway to control the programming phase. Consequently, it is actually this one that in practice replace all counters with a step value less than two.

To illustrate this, we give an example with a the 16-point fully-streaming NTT, with a streaming width of 4 (Figure 3.7). The input twiddle flow is:

in_tw_0 → 0 2 4 6
in_tw_1 → 1 3 5 7

The counter and bunch's index associated to each register and each memory of the twiddle banks are:

- Reg (1,1)→ counter(offset = 2, step = 0); index : 0 ⇒ Extracts [4]
- Mem (2,0)→ counter(offset = 0, step = 2); index : 0 ⇒ Extracts [0,4]

[I] n, r, and w being powers of two.

- Mem (2,1)→ counter(offset = 1, step = 2); index : 0 ⇒ Extracts [2,6]
- Mem (3,0)→ counter(offset = 0, step = 1); index : 0 ⇒ Extracts [0,2,4,6]
- Mem (3,1)→ counter(offset = 0, step = 1); index : 1 ⇒ Extracts [1,3,5,7]

The result of the extraction is what is expected in each memory element of a twiddle bank.

Feeding the data path. The mechanism instantiated in the IDP module that allows feeding the appropriate **FFValues** to each data path stages is similar to the selection of the programmed twiddle bank.

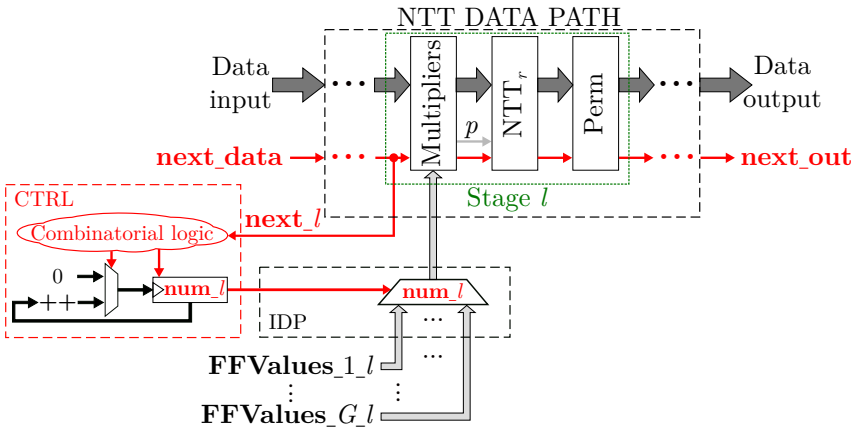


Figure 3.14: Selection of finite-field values for a stage l.

This mechanism is illustrated in Figure 3.14. The twiddle bank from which the **FFValues**_l are read is selected according to the control register **num**_l located in the CTRL module. The arrival of a different finite field in the data path flow is signaled by a **next** signal. On its reception, the cyclic update of the corresponding **num** register in the CTRL module is performed .

The **next** signals are also responsible of the re-synchronization of the *read address* generators concatenated in the GA module. This is not shown in Figure 3.14 for simplicity.

In this section, we have presented the principle of our circular handling of twiddle factor sets. This solution allows to define a fully-streaming NTT circuit that is able to handle multiple RNS channel without loss of performances. In the next section, the hardware cost overhead of our solution is studied.

3.4 Synthesis results and comparisons

In this section, the cost of our circular handling of twiddle factor sets is studied. In a first time, the cost of our solution is compared to the cost of a single-field NTT. In a second time, we take into account the context of RNS representation and compare our solution with another approach from the literature.

The integration of our proposal in SPIRAL for automatic generation of the NTT data paths has not been fully completed yet. Hence, we can only present the synthesis results for

the twiddle path. We developed a python script that has automatized the generation of the twiddle factor paths for radix-2 fully-streaming NTT ($r = 2$). Synthesis have been performed with the default mode of Xilinx Vivado's synthesizer (2018.1). The targeted FPGA is a Virtex 7 xc7vx690t from Xilinx.

3.4.1 Overhead of the twiddle path

In Table 3.1, we have detailed the differences in terms of resource utilization between our twiddle path and the cost of the twiddle handling for a single-field NTT. The resource cost of the twiddle handling of a single-field NTT is roughly the cost of one of our twiddle bank.

Table 3.1: Resource utilization of twiddle paths for multi-field NTT compared to single-field NTT. The resource utilization in the latter case is considered to be that of a single twiddle bank (TWB). The resource utilization is expressed w.r.t. the resource available on a Xilinx Virtex 7 xc7vx690t.

(n, w, s, G)	Res.	Our Twiddle Path						%	**TWB**
		%	**Total**	STWB	PRG	CTRL	GA		
$(2^{12}, 2, 30, 4)$	LUTs	**0.59**	**2,567**	2,043	355	117	51	**0.13**	**562**
	FFs	**0.45**	**3,912**	3,512	237	97	66	**0.11**	**944**
	BRAMs	**1.9**	**28**	28	–	–	–	**0.48**	**7**
$(2^{14}, 2, 30, 4)$	LUTs	**0.71**	**3,055**	2,350	498	137	70	**0.15**	**658**
	FFs	**0.52**	**4,495**	3,968	289	115	91	**0.13**	**1,083**
	BRAMs	**5.17**	**76**	76	–	–	–	**1.29**	**19**
$(2^{14}, 8, 30, 4)$	LUTs	**1.83**	**7,950**	6,117	1,577	202	51	**0.37**	**1,581**
	FFs	**1.7**	**14,658**	13,568	813	115	66	**0.40**	**3,458**
	BRAMs	**7.62**	**112**	112	–	–	–	**1.9**	**28**
$(2^{14}, 8, 46, 4)$	LUTs	**2.5**	**10,826**	8,997	1,573	202	51	**0.53**	**2,301**
	FFs	**2.47**	**21,350**	20,064	993	115	66	**0.59**	**5,082**
	BRAMs	**16.3**	**240**	240	–	–	–	**4.08**	**60**

The overhead of our circular-handling of twiddle factor set is mainly dependent of the number of instantiated twiddle banks G. The surrounding parts have a relatively small impact on resources utilization. For the parameter sets considered in this thesis, G was always equal to four, and it is very unlikely become larger in our context. Indeed, for G to get larger, the ratio $T = n/w$ has to be reduced. If one increases w, he has to face bandwidth and hardware cost issues (see Section 5.3), and the minimal n for RLWE security is chosen above 2048 in practice.

The overhead of our twiddle path is not very important compared to the advantages it brings in our context. This is highlighted in the next subsection.

3.4.2 Comparisons with a straightforward storage of twiddle factor sets

With the RNS/NTT coupled approach for polynomial multiplication, a basic handling of multiple twiddle factor sets could be very costly. Our twiddle handling allows a designer to

Table 3.2: Resource utilization for a local storage compared to our on-the-fly handling of twiddle factor sets.

Parameters						Storage as in [81]		**Our Twiddle Path**	
L	n	S_q	s	$k+k'$	w	MB	BRAM	MB	**BRAM(%)**
1	2^{11}	54		5		0.31	25	0.25	**20 (-20 %)**
5	2^{12}	108		8		0.98	56	0.49	**28 (-56 %)**
10	2^{13}	216	30	16	2	3.93	176	0.98	**44 (-75 %)**
20	2^{14}	432		30		14.75	570	1.97	**76 (-87 %)**
30	2^{15}	594		41		40.30	1,394	3.93	**136 (-90 %)**
					2		570		**76 (-87 %)**
20	2^{14}	432	30	30	4	14.75	660	1.97	**88 (-87 %)**
					8		840		**112 (-87 %)**
					16		1,200		**160 (-87 %)**
			30	30		14.75	570	1.97	**76 (-87 %)**
			41	22		14.78	693	2.69	**126 (-82 %)**
20	2^{14}	432	51	18	2	15.04	594	3.34	**132 (-78 %)**
			58	16		15.20	592	3.80	**148 (-75 %)**
			62	15		15.24	585	4.06	**156 (-73 %)**

consider the twiddle factors as inputs for an NTT-based hardware accelerator. Consequently one does not have to locally store all the twiddle factor sets (one for each RNS channel) in order to implement the coupled approach.

To highlight the benefit of our strategy, we compare our twiddle factor handling with a local storage strategy as in the work of Cousins et al. [81]. We remind that they have also implemented a fully-streaming NTT but they have chosen to store the NTT twiddle factors in ROMs filled up at compile time.

Note that we do not directly compare with their synthesis results as they are not accessible in their paper. What we did is a projection of the BRAM utilization based upon the utilization of one of our twiddle bank multiply by the number twiddle factor sets.

In Table 3.2, the advantage of using the circularly-buffered handling of twiddle factor sets rather than full local storage appears clearly. Our on-the-the fly handling of twiddle factor sets reduces from 20 % up to 90 % the BRAMs utilization in the context of FV acceleration. This allows our multi-field NTTs to address larger FV's parameter sets.

3.5 Conclusion

In this chapter we have explored an approach for the definition of an efficient NTT-based polynomial multiplier for FV acceleration. Our choices to explore the adaptation of the SPIRAL DFT generator in our context is motivated by the presentation of its design space exploration capability. This theoretically allows us to consider very large degree n for FV accelerations. But the twiddle factor handling system has to be compatible with the RNS representation in order to not losse execution performance due to the reprogramming of twiddle factor sets.

The main contribution of this chapter is the definition of a fully-streaming multi-field NTT,

without loss of performances and without major hardware cost overhead. This is achieve by circularly-buffering the multiple twiddle factor sets. Another contribution is the hint that the automatic generation of such architectures is possible for generalized NTT circuits. In particular, this work has shown the feasibility of automatic generation of fully-streaming multi-field NTTs. We believe that there are no major constraints to the adaptation of our solution to other type of architecture generated by SPIRAL, and more generally to any types of NTT circuits.

With regard to the hardware acceleration of FV, our twiddle factor handling allows the design of high throughput NTT-based residue polynomial multipliers, while having an hardware cost independent of the RNS basis sizes. Nevertheless, our solution alone only postpones the storage of twiddle factors. Thus, a decrease in performance due to heavier communications has to be considered. Hopefully, our twiddle factor generator presented in the next chapter propose a solution to this problem.

Chapter 4

On-the-fly computation of NTT twiddle factors

In previous chapter the strategy of on-the-fly generation of twiddle factor sets has been motivated. This chapter presents our solution for this generation. The problematic is reminded with some in-depth on the throughput requirement in our context. Then, a generic design that achieve the throughput requirement is presented. This is followed by the proposal of an optimized recurrence relationship to generate a single twiddle set. Finally, the hardware cost and the benefits of this generator are presented.

4.1 On the issue of generating multiple twiddle factor sets

Our application context considers the acceleration of the full RNS variant of FV. In particular, we are currently focused on the definition of an efficient hardware implementation of NTT-based Residue Polynomial Multiplier (RPM).

In the first subsection, the expected pre-computed values for NTT-based RPMs are reminded. It shows that there is no fundamental difference in the generation of twiddle factors for Padded Convolution (PC) and Negative Wrapped Convolution (NWC) in practice. Then the distinction between the throughput requirement and the choice of a recurrence relationship is expressed.

4.1.1 Reminders on twiddle factors

The computation of a n-point NTT over the finite-field $\mathbb{Z}_{p_i}$ requires the choice of a primitive n-th root of unity $\omega_i \in \mathbb{Z}_{p_i}$. The transformation of the n-sequence $\mathbf{a} \in \mathbb{Z}_{p_i}^n$ is also a n-sequence $\mathbf{A}$ such that:

$$\text{For all } j \text{ in } \{0, ..., n-1\},\ A_j = \sum_{k=0}^{n-1} \omega_i^{jk} a_k \bmod p_i. \tag{4.1}$$

The inverse transformation is defined by:

$$\text{For all } k \text{ in } \{0, ..., n-1\},\ a_k = n_i^{-1} \sum_{j=0}^{n-1} \omega_i^{-kj} A_j \bmod p_i. \tag{4.2}$$

The values ω_i^{-kj} and n_i^{-1} are the multiplicative inverse of ω_i^{kj} and n over $\mathbb{Z}_{p_i}$.

In the case of a RPM through PC, the polynomials of degree n are padded with zeros up to N elements (N a power of two in practice such that $N \geq 2n$). The convolution is then performed on N-points sequences. It then requires the twiddle factor set $\{\omega_i^k\}_{i=0}^{N/2-1}$ for NTT and $\{\omega_i^{-k}\}_{i=0}^{N/2-1}$ for inverse NTT.

For RPM through NWC, the convolution is directly performed on n-points, but requires extra pre-computed values. Namely, a primitive n-th root of -1 over $\mathbb{Z}_{p_i}$ noted ψ_i that define weight vectors $\Psi_i = (\psi_i^j)_{0 \leq j < n}$ and $\Psi_i^{-1} = (\psi_i^{-j})_{0 \leq j < n}$. The NWC is then a weighted convolution on n-points, and the actual twiddle factors for NTT and inverse NTT are subset of Ψ_i and Ψ_i^{-1}.

We indifferently call twiddle factors the pre-computed values required to computes these operations. In both PC and NWC, we have to generate 2^k-sequence of powers of a finite-field value. Consequently, our discussion will only consider this generic expression of the problem.

In addition to this, our context imposes the generation of multiple sequences in data flow while respecting a given throughput.

4.1.2 Throughput requirement

Our exploration of fully-streaming approach requires a twiddle factor set generation with the same throughput as the data path of the considered convolution (PC or NWC). As seen in the previous chapter, the throughput of the data path is only dependent on the transform size n and the streaming width w. Namely, $T = n/w$ cycles separates two consecutive RNS channels in the data path. Thus, our generation module has to output a new twiddle factor set of n elements every $T = n/w$ cycles. Consequently, the solution to respect the throughput is to be able to generate exactly w elements per cycle. Being able to generate more than w per cycle is considered sub-optimized as it would require having more computing element than what is strictly necessary.

The difficulties of this generation come both from the dependence between the elements of the sequence of powers and from the latency of the modular multipliers that computes the elements. Some inevitable bubbles occur in the generation of a single twiddle factor set. Hence, generating one set after the other does not achieve the required throughput.

The next section details our proposed generic architecture solution that achieve the required throughput. Basically, this is done by overlapping multiple set generations. But beforehand, let us consider the problematic of choosing a recurrence relationship to generate a single twiddle factor set.

4.1.3 Recurrence relationship for a single set generation

The choice of the recurrence relationship for the generation of a twiddle factor set has a major influence on the hardware efficiency of the solution. Indeed, there is a compromise to find between the latency of a generation and the intermediate storage required to perform the generation.

The choice of this recurrence relationship is equivalent to the search for an overlap in a dependency graph. Different solutions can be found regarding different constraints. In our case, we propose an optimized solution that makes intermediate storage independent of the size of the sequence to generate. This solution is expressed after having introduced straightforward sub-optimized solutions in our context. This is detailed in section 4.3.

Now that both the throughput problematic and the recurrence relationship problematic have been settled, our solutions are detailed in two different sections.

4.2 Data-flow oriented twiddle factor set generator

In this section is presented our solution to achieve the desired throughput of multiple twiddle factor sets. The principle is that when there are inevitable bubbles in the generation of a set, the generator fills these bubbles with calculations from other set's generations ready to be performed. This results in a mixed set output sequence from which each twiddle factor set has to be sorted out.

4.2.1 Design overview

In our generic design solution, the generation of the sequence is separated from the sorting process. Consequently, the generator is composed of two modules as presented in Figure 4.1. The COMPUTE module is responsible for the generation, and the SORT module for sorting the different sequences. The generator handles up to H different twiddle set generations at the same time.

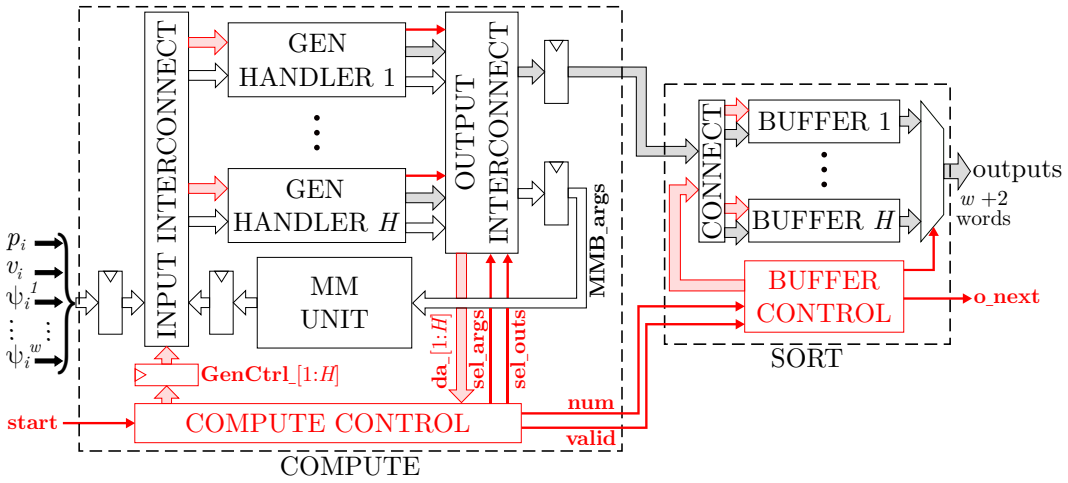


Figure 4.1: Generation of twiddles (GEN TW) flow.

From a high level point of view, it is required to compute n elements in T cycles, so if the generator outputs w elements per cycle the required throughput is achieved. Hence, our generator instantiates exactly w modular multipliers (MM UNIT) and the rest of the design take care of feeding these computing units and sorting the results.

The COMPUTE module schedules the different set generations on the computing resource MM UNIT. The outputs of the COMPUTE module are associated with a **num** and a **valid** signals, which allow the proper sorting of the twiddle factor sets.

Each set generation is associated to a specific GEN HANDLER. Each GEN HANDLER organizes the intermediate storage and the computations according to the chosen recurrence relationship for a single set generation. More details are given in section 4.3.

The COMPUTE CONTROL module instantiates the scheduling process that guarantee a sequential access of the different set generation to the MM BANK.

The SORT module instantiates H different buffers in which the twiddle factor sets are sorted. When a twiddle set generation is completed, the BUFFER CONTROL initiates the output of the n-sequence, w elements per cycle. The concerned GEN HANDLER and BUFFER are then re-used for a new twiddle set generation.

The number H of simultaneous twiddle set generations depends on the latency of a single generation. Thus, it is dependent on the recurrence relationship and on the latency of the pipelined modular multipliers. Indeed, both influence the number of bubbles in a single set generation that has to be filled up with other generations. Nevertheless, H is chosen as the smallest value necessary to reach the desired throughput. That is $H = \lceil Lat\_GENTW_{max}/T \rceil + 1$, with Lat_GENTW_{max} being the largest possible latency of the GEN TW module.

Due to the scheduling implemented in COMPUTE CONTROL a twiddle factor set generation has its latency dependent on the workload of the generator. The maximum latency Lat_GENTW_{max} occurs when the GEN TW module is used at maximum capacity: one new set generation request every T cycles. The minimum latency Lat_GENTW_{min} occurs when the GEN TW module is used at minimum capacity: one single set generation.

We have given an overview of our twiddle factor set generator. In the next subsections, the different sub-modules are detailed.

4.2.2 Computing the twiddle sets

The COMPUTE module handles the scheduling of the required twiddle factor sets. This module is composed of H GEN HANDLER modules. Each takes care of a specific twiddle factor set generation. The COMPUTE CONTROL module organizes the sequential access of the GEN HANDLERs to the MM UNIT by scheduling them according to a priority rule.

Modular multipliers unit. The purpose of the MM UNIT module is to compute the next w elements of the different twiddle set generations. This module has no control unit as it simply executes the computation of elements accordingly to its current inputs.

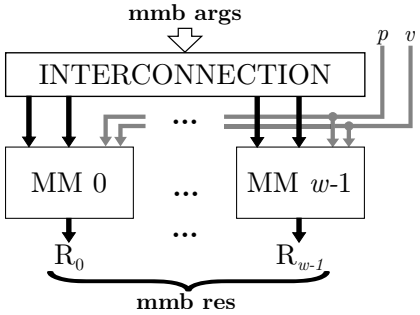


Figure 4.2: MM UNIT principle.

The Figure 4.2 presents the principle of the MM UNIT unit. All the modular multipliers are configured by the (p, v) pair on input specifying the finite-field of the current arguments **mmb args**. The number of arguments and their connection to the modular multipliers depend on the chosen recurrence relationship. The results (**mmb res**) feed the generation handlers and the COMPUTE CONTROL is responsible of updating the **valid** control signal of the appropriate GEN HANDLER.

Generation handler. Each generation is started by giving some seed elements along with the associated (p_i, v_i) pair. Namely, the prime defining the finite-field $\mathbb{Z}_{p_i}$ and its reciprocal for efficient modular multiplication. In our case, we consider w elements to stay consistent with the required throughput.

A GEN HANDLER module implements the chosen recurrence relationship upon the reception of the initial seed elements. This requires a dynamic storage of some intermediate results, and preparation of the next computations. The general principle is described here.

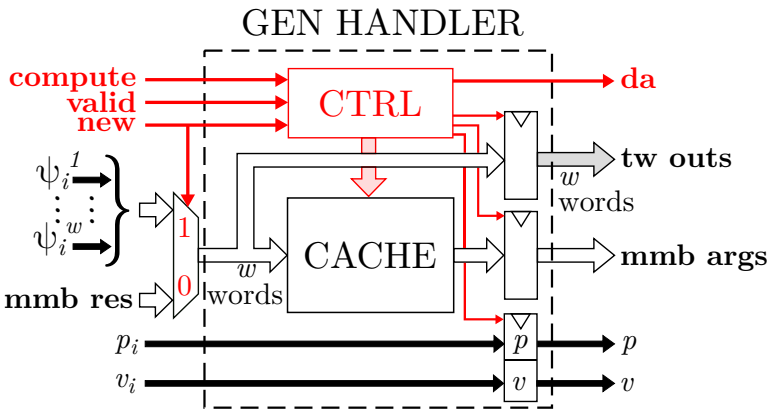


Figure 4.3: Generation handler general principle.

As illustrated in Figure 4.3, a GEN HANDLER merely consists in a cache memory and two main outputs registers. The storage elements are controlled by a control module under the influence of three control signals and some internal counters. These control signals **compute**,**valid** and **new** are generated by COMPUTE CONTROL from Figure 4.1.

The signal **new** indicates that the current inputs are seed elements for a new twiddle set generation. This signal implies the local storage of the finite-field specific pair (p_i, v_i), and the re-initialization of the CTRL module.

The signal **compute** indicates that COMPUTE CONTROL has scheduled this specific GEN HANDLER to use the MM UNIT. Consequently, the CACHE is accessed to prepare the arguments (**mmb args**) of the next computation of the twiddle set. In order to be scheduled, GEN HANDLER has to indicate whenever it is ready to perform new computations for its set generation. This is done by setting the **da** signal to logic high.

Finally, the signal **valid** indicates that the current w inputs are valid elements, of the current set generation. These elements have been produced by MM UNIT after the GEN HANDLER has been scheduled by COMPUTE CONTROL. This **valid** signal implies the preparation of the **tw outs** register that outputs the following w elements of the sequence.

The CTRL sub-module keeps trace of the current state of the generation. This state is managed using three different counters: # *data* # *store*, and # *compute*. The first keeps trace of the number of elements already generated, the second the number of elements stored in CACHE, and the third the number of times GEN HANDLER has been scheduled by COMPUTE CONTROL.

These pieces of information are required for the translation of the input control signals into a cache management and an output register management consistent with the chosen recurrence relationship for a set generation. In section 4.3, this generation handler principle is detailed with an example.

Computation control. The purpose of COMPUTE CONTROL is to schedule a sequential access of the different twiddle set generations to the MM UNIT. The desired result is that the MM UNIT is constantly performing new computations, thus the required throughput is achieved.

In our solution, the scheduling process chooses one GEN HANDLER, among those ready to be scheduled, accordingly to a decreasing priority with the advancement of their twiddle set generation. If a new set generation is starting, it preempts any computation for that cycle, and becomes the highest priority generation.

In practice, if each start of set generation is separated by at least T cycles, each set will be completed, and their order of completion will be the same as their starting order[I]. As a consequence, only the knowledge of the highest priority generation at a given time is necessary to know the order of priority of all current generations. The respect of the T cycles interval between two generation starts is considered to be under the responsibility of the surrounding system.

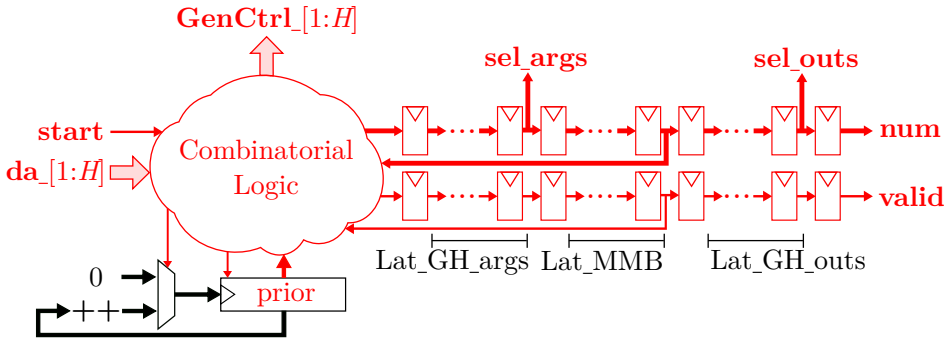


Figure 4.4: COMPUTE CONTROL principle.

Figure 4.4 illustrates the resulting module that essentially consists in a pipelined propagation of the scheduling results. The **prior** register indicates the generation currently with the highest priority. This register is cyclically updated in $1, ..., H$ as new generations are requested (reception of the **start** signal). The signals **da_[1:H]** are coming from the different GEN HANDLER [1:H]. They indicate the GEN HANDLERS that are currently able to perform new computations, and thus which are eligible in the current scheduling session.

[I]The number of modular multipliers in MM UNIT is chosen to exactly reach the requested throughput T. If the start of a set generation do not respect this throughput, our scheduling rule will cause the generations to preempt each others and some deadlocks occur.

The scheduling session may have three different outcomes: one computation is scheduled, no computation is scheduled, or a new generation is started.

In the first case, the index of the GEN HANDLER elected is propagated into the **num** shift register and a logic high signal is propagated in the **valid** shift register. The **compute** signal of the appropriate GEN HANDLER is set to logic high.

In the second case, a logic low signal is propagated in the **valid** shift register and no GEN HANDLER **compute** signal is set.

In the third case, the **prior** signal is updated and the index of the GEN HANDLER receiving the responsibility of the new generation is propagated into the **num** shift register. A logic high signal is also propagated in the **valid** shift register.

The other control signals driving the COMPUTE module behavior are updated accordingly to the propagation of the scheduling results into the two shift registers. The depth of these shift registers depends on the recurrence relationship and on the latency of the modular multipliers. At the end of the two shift registers, the **num** and **valid** signals are propagated to the SORT MODULE presented in the next subsection.

All the principal sub-modules of COMPUTE have been presented. The INPUT INTERCONNECT and the OUTPUT INTERCONNECT are just simple routing modules that embedding multiplexers and registers for the realization of the expected functionality.

The next section presents the sorting of the different set generations.

4.2.3 Sorting the twiddle sets

The SORT module sorts the different twiddle factor sets from the mixed output sequence of the COMPUTE module. To do so, it temporarily stores the sets in H different buffers, each associated to a GEN HANDLER. When a generation is completed, it outputs the n-sequence in a single flow.

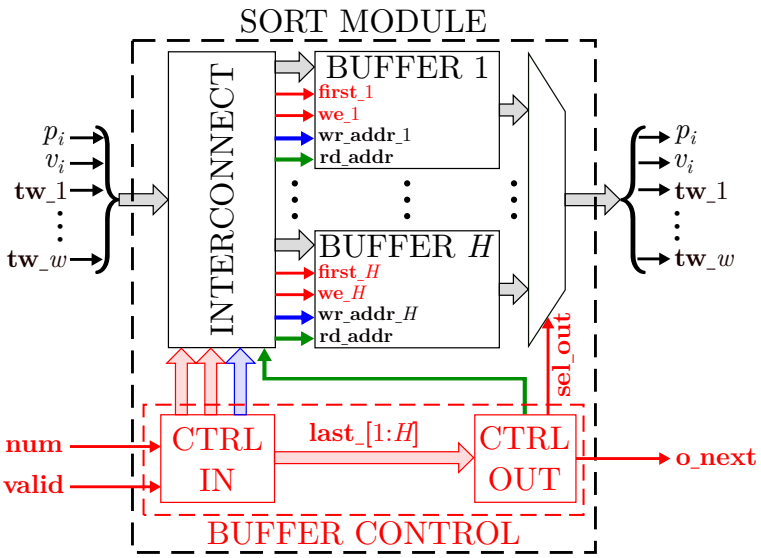


Figure 4.5: SORT module.

Figure 4.5 presents a detailed illustration of the SORT module. Each BUFFER is actually a bunch of w memories, and each memory is associated to one of the w streaming ways.

The BUFFER CONTROL module is decomposed into two sub-modules. The first one (CTRL IN) updates the INPUT INTERCONNECT and generates the write-address and write-enable signals for the currently fed BUFFER. The second one is responsible for the output of the n-sequences once they are completed.

The details of the different sub-modules are described in the following paragraphs.

Buffer module. A BUFFER module is composed of w single port memories and a pair of registers to store the $(p_i,\ v_i)$ values. It is controlled by four signals: a read-address, a write-address, a write-enable and a **first** signal.

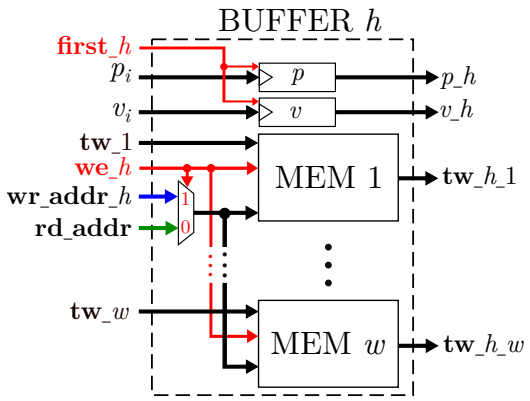


Figure 4.6: BUFFER module.

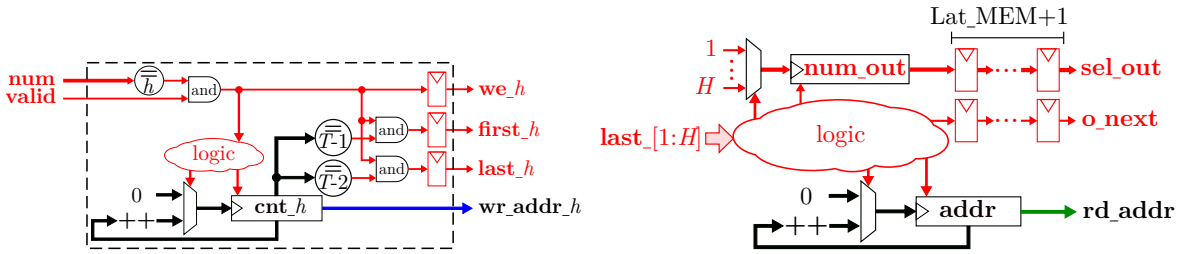
The Figure 4.6 illustrates the internal structure of a BUFFER module. Each memory of the BUFFER module is a depth T single port memory and stores the twiddles from the corresponding streaming way. All the memories are accessed for read or for write simultaneously and the **we** signal is used to select between the address signals.

The $(p_i,\ v_i)$ pair is stored when the first elements of the twiddle factor set are inputted. This is signaled by the **first** signal generated by the BUFFER CONTROL module.

Buffer control. The BUFFER CONTROL module is organized into two sub-modules. One is responsible for sorting the elements coming from the COMPUTE module, and the other handle the output of a twiddle factor set as soon as it is completely sorted.

The CTRL IN sub-module is illustrated Figure 4.7a. For each buffer, a counter is cyclically updated between 0 and $T-1$ whenever a new valid bunch of w elements is generated by the corresponding GEN HANDLER. The current value of the counter is used as write address for the BUFFER, and the **we** signal is simply generated from **valid** and **num**.

When the counter is updated to 0, the **first** signal is issued, specifying that it is a new twiddle set. Similarly, when the counter is updated to $T-1$, a **last** signal is set to specify that a twiddle set is ready to be outputted. The CTRL OUT sub-module (Figure 4.7b) receives this signal and handles the output of the twiddle set.



(a) CTRL IN sub-module.

(b) CTRL OUT sub-module.

Figure 4.7: The two sub-modules of the SORT CTRL module.

The CTRL OUT sub-module is illustrated Figure 4.7b. Whenever a **last** signal is received from the CTRL IN sub-module, the corresponding buffer is ready to be outputted. Nevertheless, the different **last** signals may fire with different intervals between each other. This is due to the fluctuating latency of the COMPUTE module with respect to its current workload. Consequently, the CTRL OUT sub-module has to store in FIFO order the different output requests, and treats them one after the other.

As long as there is a twiddle factor set ready to be outputted, a read address generator is incremented from 0 to $T-1$. The corresponding buffer's index is propagated as the **sel_out** signal, controlling the output multiplexer of the SORT module. The **o_next** signal is only set for one cycle before the actual output of the twiddle set.

4.2.4 Remarks

The overview of a generic twiddle factor generator has been given in this section. The principle is to fill up the bubbles of a set generation by overlapping different set generations over time. We choose to instantiate as few computing units as possible with exactly w modular multipliers. The different twiddle set generations access this bank sequentially according to a priority scheduling rule.

The SORT module is only dependent on the size of the sequence to generate n and the streaming width required w. But the choice to instantiate depth T memories in the BUFFER modules could be furthermore improved. Indeed, the size of these buffers could be only dependent on the number of bubbles in the generation of a set. The actual number of bubbles is far lower than T in our application context, but knowing its exact value is a bit tricky. Due to time constraints, we did not further explore this improvement.

The COMPUTE module is also dependent on the chosen recurrence relationship generating a twiddle factor set. In particular, it derives the concrete implementations of the GEN HANDLERs and the MM UNIT. Finally, the number H of simultaneous twiddle set generation depends on the overall latency of the generator module with respect to the required throughput. When T is large in front of a modular multiplier latency (which is true for the considered parameter sets), $H = 3$ is sufficient to saturate the MM UNIT with twiddle computations and achieve the required throughput.

In next section the choice of the recurrence relationship is discussed and two different choices are presented. In particular, we show that the second is optimized in our context.

4.3 Choice of a recurrence relationship

This section presents the problematic of generating n-sized sequence of powers of a number. Two recurrence relationship and their specific implementations in our generator are presented. This presentation uses a graph formalism to represent the problem and the proposed generation heuristic.

4.3.1 General problem presentation

The purpose of our generator is to generate multiple n-sized power sequences of numbers with a throughput of $T = n/w$. In other words, it has to generate a new sequence $\{a_i^k\}_{k=0}^{n-1}$ $(i \in \mathbb{N})$ every T cycles. In our case both n and w are powers of two, and the elements a_i are parts of some finite-fields defined by some prime p_i $(\mathbb{Z}_{p_i})$.

In general, the generation of a sequence $\{a^k\}_{k=0}^{n-1}$ requires at least the knowledge of the initial number a. Here we consider that we have been given w initial elements at the beginning of a sequence generation. These initial elements are then used to compute further elements which themselves make the computation of further elements possible. This principle is applied until the n-sequence is computed. The problem here is that there exist a large number of recurrence relationships to generate a n-sequence of powers of a number.

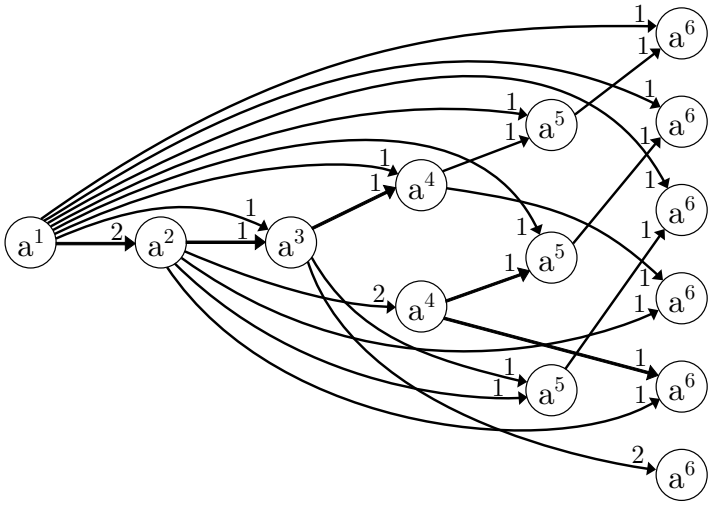
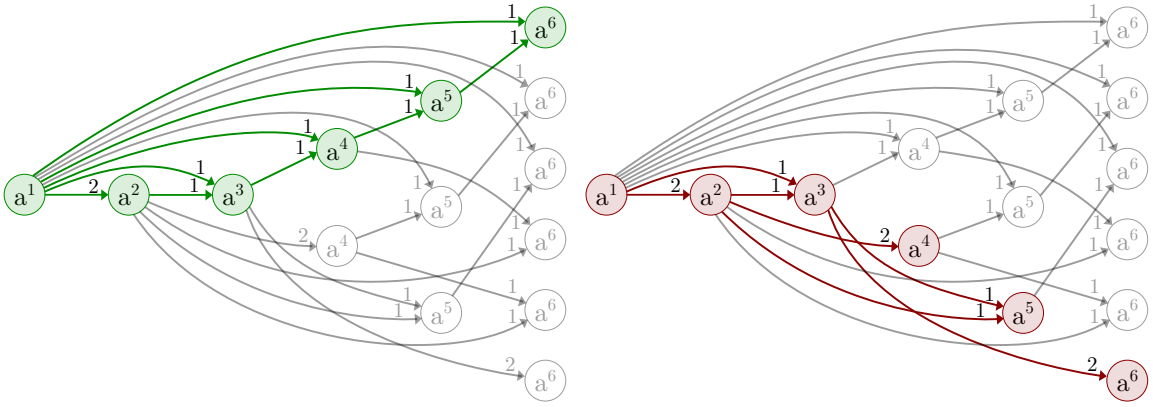


Figure 4.8: Dependency graph of the powers of a. Here $n = 6$ and only multiplication operations with 2 operands are considered.

In Figure 4.8, a graph formalism is used to represent the dependencies between the elements of the 6-sequence of powers of an element a.

Each node of the graph represents one of the powers to be generated. The oriented arcs identify the parent(s) of each node, and the arc's weight represents the kinship factor of the two nodes. Considering here only multiplication with two operands, the input kinship factor of each node is exactly 2, and the weight of each arc is 1 or 2. There are as many nodes for the k-th element as there are ways to calculate it. The number of nodes is then exponential with k, but generating the sequence is to choose an overlap of the dependency graph. We call overlap here a sub-graph containing only once each element of $\{a^k\}_{k=0}^{n-1}$.

Many overlaps are possible, but some are more convenient than others regarding practical implementation of the considered generation (overall latency, intermediate storage required, complexity of computing structure, ...). In our case, we want methods of finding overlaps that are expressed with a recurrence relationship. This requirement guarantees that the implementation in hardware of the generation is simple.



(a) Overlap minimizing intermediate storage and number of computing units

(b) Overlap minimizing latency

Figure 4.9: Examples of overlaps given by two different recurrence relationships. Examples for $n = 6$.

The Figure 4.9 illustrates two examples of overlaps generated obtained with a recurrence relationship. The first one (Figure 4.9a) is obtained with repetitive multiplications with the first element $\mathbf{a}$.

$$\begin{cases} U_0 = 1 \\ U_1 = a \\ U_{k+1} = U_1 \times U_k \end{cases} , \forall\ k \in [1\ ;\ n-2] \cap \mathbb{N}$$

This recurrence relationship minimizes the intermediate storage needed to generate the n-sequence. Indeed, only the first element a needs to be kept during the overall computation. This is the straightforward computation of the geometric sequence of common ratio a.

The drawback of this solution is that the generation has the largest latency possible. Indeed, when considering multipliers with a latency larger than 1, each new elements has to wait for the generation of the previous one. This result in a overall latency of $(n-1)$ times the latency of the computing unit.

The second overlap in Figure 4.9b is obtained with a solution that focuses on being able to compute further elements as soon as possible. Hence, considering 2-operands multiplication, the recurrence relationship generating the overlap is:

$$\begin{cases} U_0 = 1 \\ U_1 = a \\ U_{2k} = U_k \times U_k \\ U_{2k+1} = U_k \times U_{k+1} \end{cases} , \forall\ k \in [1\ ;\ n/2-1] \cap \mathbb{N}$$

This recurrence requires the ability to dynamically store (and potentially overwrite) elements in memory close to computing units. At established regime, it results in a proportion of two generated elements for one already known. This means that only the first $n/2$ first elements of the sequence has to be known to be able to compute the further $n/2$ elements. Moreover, if the oldest stored intermediate elements are dynamically overwritten with new ones during the sequence computatio. Hence, the required intermediate storage space is $O(n/4)$ elements.

The second recurrence relationship is estimated more interesting in our context. Minimizing the latency of the generation makes us instantiate less simultaneous set generation H. But this solution has an important drawback, that is the required intermediate storage of $O(n/4)$ elements. This is problematic for large FV's parameter sets. Furthermore, the ability of generating the element as soon as possible is of no use in practice. Indeed, the fixed number of computing resources implies a maximum of w elements generated per cycle.

It is actually possible in our case to find a recurrence relationship that minimizes both the latency and the intermediate storage. Its expression is dependent on the number of computing units and their latency.

The next subsection presents this optimized recurrence relationship.

4.3.2 An optimized recurrence relationship

We remind that we consider w initial elements at the beginning of a sequence generation, and that the number of computing units is exactly w. Thus, a n-sequence is then composed of $T = n/w$ bunch of w elements each, and two consecutive bunches have their powers only increased by w. Hence, two different bunches have necessarily their powers separated by a multiple of w (illustrated in Figure 4.10). Consequently, only some powers multiple of w plus one bunch have to be accessible to be able to compute further bunches. The question is then to find the number of powers multiple of w that are strictly necessary to achieve the lowest latency in the generation of the T bunches.

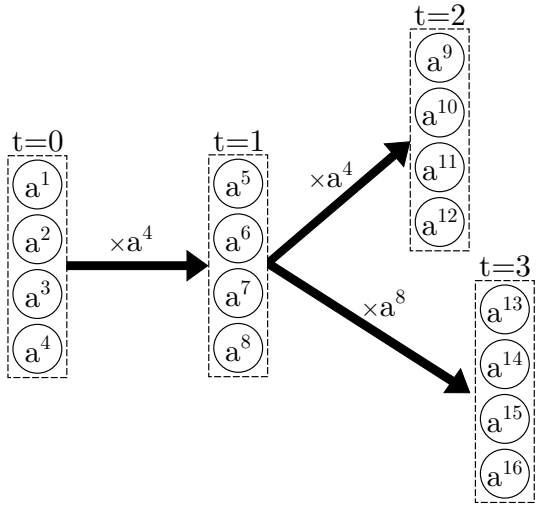


Figure 4.10: Bunch-wise generation of the n-sequence. Example for $n = 16$ and $w = 4$.

This is easy to express once the recurrence relation that links the bunches together is formalized.

Our recurrence relationship. We expressed as follow the optimized recurrence relationship over which our generation is built:

$$\begin{cases} t_0 = \begin{cases} a^1 \\ \dots \\ a^w \end{cases} \\ f_i = a^{iw} & , \ \forall\ i \in [1, n/w - 1] \\ t_r = t_k \odot f_{r-k-1} & , \ \forall\ r, k \in [1, n/w - 1] \end{cases}$$

From the initial knowledge of the bunch t_0 we already know the first factor $f_1 = a^w$. A new bunch computation is the point-wise multiplication of a known bunch with a known factor. If we currently have access to the k-th bunch, the computation of the r-th bunch ($r > k$) requires the knowledge of the factor f_i such that $i = r - k - 1$.

In addition to this recurrence relationship, we make an additional design choice. Namely, we always compute the r-th bunch with the last received bunch k. Consequently, the value $i = r-k-1$ is upper bounded (noted i_{max}) by the latency between a compute request and the reception of the generated bunch. This latency is dominated by the latency of the modular multipliers which is mainly dependent on the size of the handled elements. For instance, our pipelined modular multipliers of subsection 3.2.3 have a latency of 21 cycles for 30-bit elements and 57 for 62-bit elements. Consequently, if we store only i_{max} factors (typically $<$ 60), we are able to start at any time, and under any preemption circumstances, a continuous flow of computation of further bunches.

Now, we have an intermediate storage independent of n while having the minimal possible latency for the sequence generation with an initial knowledge of w elements. In the next subsection we present how it is instantiated in our generic twiddle set generator.

4.3.3 Adapting the generic design

To adapt our generic twiddle generator to the chosen recurrence relationship, we only need to specify the MM UNIT input interconnection, and the cache structure of the GEN HANDLER modules. The MM UNIT argument connection is straightforward and presented in Figure 4.11.

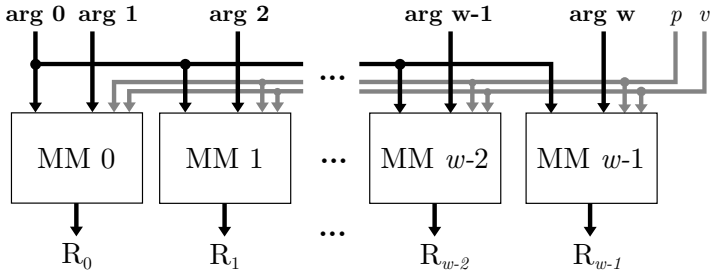


Figure 4.11: MM UNIT argument connection for the optimized solution.

The adaptation of the GEN HANDLERs is a bit more complex. Before presenting the control of the memory element, the internal structure of the cache is presented. The following description refers to the Figure 4.12. A GEN HANDLER cache implements w registers for the last received bunch, and a bank of i_{max} registers for the factors f_i. Due to our COMPUTE module structure, $i_{max} = \min(L_{MM} + 4, T - 1)$ in our case.

Now, the update rules for the cache and the output registers under the three different control events are described. In addition, the generation of the **da** signal is also made explicit.

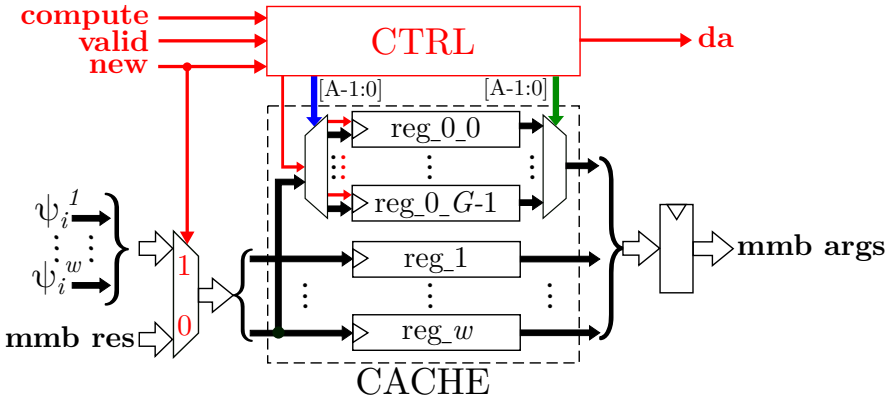


Figure 4.12: GEN HANDLER cache structure for the optimized solution.

- When new set generation is required (**new** = '1').
  The output registers **tw_out** is set with the w first value of the sequence $(a^0, a^1, ..., a^{w-1})$ (not shown in Figure 4.12). The bunch on input is stored in the cache, and its w-th element is permanently stored in the bank of registers.

- When the computation of a new bunch is scheduled (**compute** = '1').
  The last received bunch is set in the **mmb args** register along with one of the factors from the register bank. The factor chosen is dependent on the $\#compute$ and $\#data$ counter such that the recurrence relation $i = n - k - 1$ is verified.

- When a new bunch is received (**valid** = '1').
  The output registers **tw_out** receive the w next element of the sequence (not shown in Figure 4.12). The bunch on input replaces the last received bunch in the cache. If we do not have all the i_{max} factors yet ($\#store < i_{max}$), the w-th element of the bunch on input is stored in the register bank.
  A special case has to be treated: the first **valid** signal after a new set generation initialization does not actually notify a new bunch. It results from the COMPUTE CONTROL pipeline that has integrated the initial bunch. Consequently, nothing has to be done in that case as everything already happened in response to the **new** signal.

- Relation that trigger the **da** signal.
  It is dependent on the control signals and on the internal counters ($\#data$, $\#store$ and $\#compute$). It is always set when **new** is received. It is also set whenever the number of factor f_i stored and the last received bunch allows the computation of a new bunch. This is verified when $\#data + \#store > \#compute$ (considering here that input signals have immediate effects on counters so as not to complicate things). Of course, this is done only if we still have bunch to compute (i.e $\#compute < T - 1$).

All the specificities of our recurrence relationship on our generic twiddle generator have been described. In the next section the hardware cost of our generator is presented. Its positive impact in the context of NTT-based residue polynomial multiplication for FV acceleration is also quantified.

4.4 Synthesis results and comparisons

In this section, the cost of our twiddle factor set generator is studied. In a first time, the influence of each sub-module on the total hardware cost is detailed. The scalability of our solution upon the sizing parameters n, w and the size of the primes element s is presented. And finally, we compare our local generation solution with another approach to handle the twiddle factor sets. Namely, the external storage proposed by Öztürk et al. in [79].

For comparison purposes, we developed a python script for automatized generation of twiddle factor generators. Synthesis have been performed with the default mode of Xilinx Vivado's synthesizer (2018.1). For relative resource utilization, the targeted FPGA is a Virtex 7 xc7vx690t from Xilinx.

4.4.1 Study of the hardware cost

In Table 4.1, the partition of the resource utilization in our twiddle factor generator is presented for two different parameter sets.

Table 4.1: Detailed hardware cost partition of our twiddle factor generator.

(n, w, s)	Modules	Resources				
		Logic LUTs	SRLs	FFs	BRAMs	DSP48
		432,368		864,736	1,470	3600
$(2^{12}, 2, 30)$	**GEN_TW**	**2,281**	**360**	**5,530**	**12**	**22**
	+CMPT	1,967	359	4,668	0	22
	++SGH	1,343	0	3,072	0	0
	++MMU	442	356	1,086	0	22
	++CTRL	94	3	45	0	0
	+SORT	314	1	856	12	0
	++SBUF	98	0	546	12	0
	++CTRL	91	1	66	0	0
	+misc.	228	0	709	0	0
$(2^{14}, 8, 30)$	**GEN_TW**	**4,496**	**1152**	**11,504**	**48**	**88**
	+CMPT	3,739	1151	9,184	0	88
	++SGH	1,674	0	4,152	0	0
	++MMU	1,614	1,148	3,792	0	88
	++CTRL	96	3	55	0	0
	+SORT	757	1	2,296	48	0
	++SBUF	662	0	1,626	48	0
	++CTRL	95	1	66	0	0
	+misc.	370	0	1,789	0	0

The critical resources are in practice DSP and BRAM slices. These resources are respectively used in the COMPUTE module for the w modular multipliers in MM UNIT and in the SORT module for the H BUFFERs.

The resource utilization upon sizing parameters is presented in Figure 4.13. For the considered ranges of parameters, the influence of the sequences' size n to generate only impacts the number of BRAMs used. In practice, our generator could be made less sensitive to the variation of n. Our solution for the sorting module choose to wait for the entire sequence to be generated before letting it flowing out of the generator. But the sufficient buffers size is actually only dependent on the number of bubbles in the generation of a set. Hence, the required buffer size could be far lower than in the currently solution.

The influence of the streaming width w impacts the hardware consumption of every resources except the BRAMs. In practice, this parameter should not get too large due to the tremendous bandwidth requirement it imposes on the NTT data path side. Even so, the resource consumption of is still under 10% of a Virtex 7 xc7vx690t capacity for the largest w we have considered.

Finally, the influence of the size of handled elements have balanced impact on every resources. Once again, this seems a good strategy in our context to increase the size s to improve the overall performances of the FV.Mul&Relin primitive.

4.4.2 Comparisons with an external storage

We already mentioned that our twiddle factor generator completes our multi-field NTTs for the definition of NTT-based RPMs independent of the RNS basis size. In particular, this on-the-fly generation allows to avoid extra communications for bringing the twiddle factor sets locally to the accelerator. In this subsection, we quantify this advantage in the context of FV hardware acceleration. To do so, we compare our twiddle factor generation to a strategy similar to what is done in the work Öztürk et al. in 2015 [79]. Namely, the storage of all twiddle sets on external memories (from the accelerator's viewpoint).

In particular, we compare the two approaches on their memory footprint and their required bandwidth to feed a multi-field NTT. The memory footprint is defined here as the quantity of memory used by a host program to use the considered accelerator. We remind the input needs of our multi-field NTT twiddle path: $w/2$ words of s-bit per cycle. The result of the comparison is presented in Table 4.2.

In the external-storage approach the memory footprint requires $O(kn/2)$ elements of size s, compared to $O(kw/2)$ with our twiddle set generator. The memory footprint of the twiddle factors goes from 19.2 kBytes to 2.6 MBytes for the considered parameter sets. This is not critical in practice, but still, it could be avoided with local generation requiring at most 2.2 kbytes.

The stronger disadvantage of the external storage in our case is the input bandwidth requirements for bringing the twiddle in the accelerator. When considering a NTT clocked at 200MHz, storing the twiddle sets on external memories requires at least 0.75 GB/s, of communication bandwidth between the storage space and the accelerator. And this, only for the twiddle factors. With our twiddle set generator, only w words of s-bit are required every T cycles, thus saving precious bandwidth to feed the accelerator with data leading to effective speedup.

4.5 Conclusion

The general contribution of this chapter is the definition of a data flow oriented twiddle set generator that respects the throughput requirement of our streaming NTT data paths. In

Table 4.2: Memory footprint and communication bandwidth requirements for an external storage strategy and our local generation of twiddle factor sets.

Parameters						External Storage		**Local Generation**	
L	n	S_q	s	$k+k'$	w	MEM kB	BW GB/s	MEM **B**	BW **kB/s**
1	2^{11}	54		5		19.2		**80**	**2.93**
5	2^{12}	108		8		61.4		**128**	**1.46**
10	2^{13}	216	30	16	2	246	0.75	**256**	**0.73**
20	2^{14}	432		30		922		**480**	**0.37**
30	2^{15}	594		41		2,519		**656**	**0.18**
20	2^{14}	432	30	30	2	922	0.75	**480**	**0.37**
					4		1.5	**720**	**1.10**
					8		3	**1,200**	**3.66**
					16		6	**2,160**	**13.18**
20	2^{14}	432	30	30	2	922	0.75	**480**	**0.37**
			41	22		924	1	**528**	**0.5**
			51	18		940	1.3	**504**	**0.62**
			58	16		950	1.5	**512**	**0.71**
			62	15		952	1.6	**480**	**0.76**

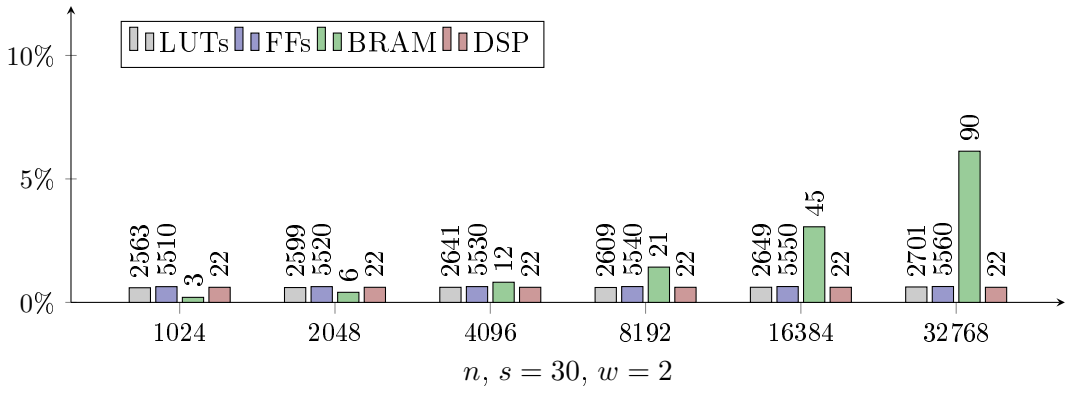
particular we propose a generic solution to achieve the desired throughput.

A more specific contribution is our proposal of recurrence relationship for optimized generations of twiddle factor sets. Our choice minimizes both the latency of a set generation and the intermediate storage space required to do so.

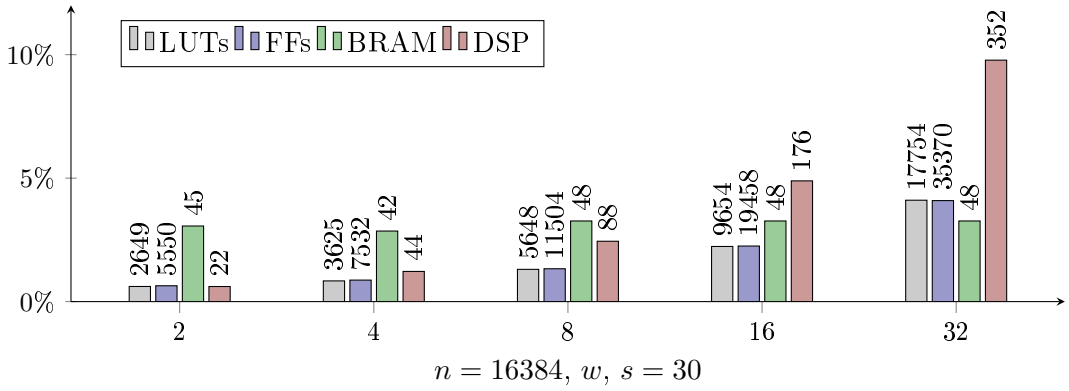
Finally, we also highlight that the SORT module could be made independent of the size n of the sequences to generate.

Along with our proposal of multi-field NTT circuits, our twiddle factor generator allows the design of high throughput NTT-based Residue Polynomial Multipliers. And this, while having a reduced influence of the RNS basis' size on the hardware cost. In the context of the hardware acceleration of FV, this opens interesting perspectives for the definition of efficient hardware acceleration.

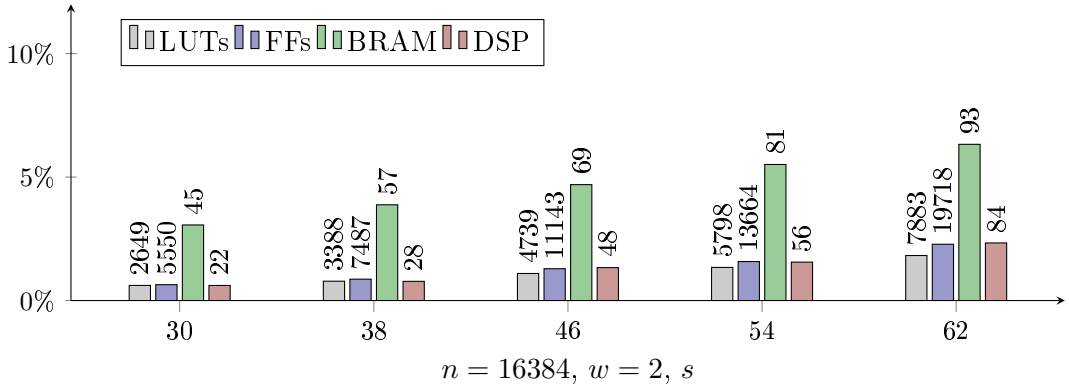
In the following chapter, we present a proposition of a computing system that use our basic hardware blocks for the hardware acceleration of the full RNS variants of FV.



(a) Influence of the NTT size n.



(b) Influence of the streaming width w.



(c) Influence of the prime size s.

Figure 4.13: Post-synthesis estimations of resource utilization for twiddle generator under the influence of sizing parameters. The relative resource utilization is estimated with respect to a virtex 7 xc7vx690t.

Chapter 5

Exploration of a hybrid strategy for the full RNS variants of FV

The analysis conducted in Chapter 2 has highlighted the necessity of accelerating both RNS specific operations and Residue Polynomial Multiplications (RPM). This came from the implementation strategy considering the coupled approach of RNS representation and NTT-based polynomial multiplications. The RNS representation tackles the complexity brought by the large modulus q, and the NTT-based polynomial multiplications tackle the complexity brought by large degree n.

In previous chapters, some basic blocks have been proposed for the hardware acceleration of the RPM operations. Dedicated hardware acceleration has been motivated by the difficulty of exploiting NTT parallelism on generic SIMD architectures due to unfriendly data-accesses.

Contrary to NTTs, the RNS specific functions embed trivial parallelism with respect to the degree n of the polynomials. This parallelism is easily exploitable with generic SIMD architectures like GPUs. In addition, having fewer different operations to accelerate on dedicated hardware significantly reduces the cost of the accelerator development.

Hence, our strategy considers the exploration of GPU accelerated RNS specific functions, and dedicated hardware acceleration for polynomial arithmetic. This result in a proposal of a hybrid computing system to accelerate the FV scheme. For prototyping, it is considered that the dedicated hardware for RPM is implemented on a FPGA, but there is also the possibility of targeting an ASIC. This is not explored in this work.

In a first section, we explore from an high-level point-of-view the communication and computation requirements for the hybrid computing system. This result in a proposal of such a system. A second section presents the perspectives for accelerating the RNS-specific functions on GPUs. Then, the third section explores the efficiency of an NWC-based RPM design that use our basic hardware blocks presented in previous chapters. Finally, this chapter concludes with the performance perspectives of our hybrid computing system proposal.

5.1 Proposal of a hybrid computing system

This section presents the motivations for a hybrid computing system for the acceleration of FV homomorphic evaluations. Our proposition considers the full RNS variant of FV proposed by Bajard et al. [4] and further improved by Halevi et al. [5]. The discussions are focused on the FV.Mul&Relin primitive, because the impact of the FV.Add primitive on execution performances is considered too small to require special consideration at this time.

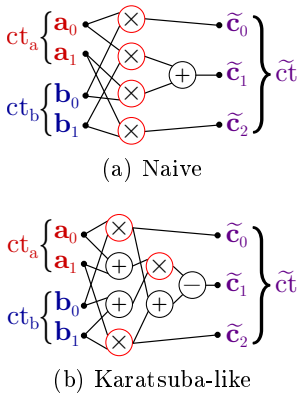
In a first subsection, the computational requirements of the system are summarized. This concludes by the computational partition on GPU and dedicated hardware. Then a high-level study of the communication requirements between the different computing units is given. Finally, the third subsection presents our proposal of system architecture.

5.1.1 Computation details for ciphertext multiplication

For the following discussions, Residue Polynomial Multiplication (RPM) refers to a multiplications over the R_{q_i}. Similarly, Residue Polynomial Addition (RPA) refers to additions over the R_{q_i}'s. Modular multiplications will be noted MM and floating-point multiplications FM. In the counting of basic operations, we do not take into account modular additions and floating-point additions. Finally, we note k the size of the initial RNS basis $\mathcal{B}$, and k' the size of the second RNS basis $\mathcal{B}'$.

The FV.Mul&Relin primitive begins with the operation of basis extension to enlarge the dynamic of the polynomial's coefficients. We remind that this is because the tensor product must be computed over R rather than R_q. If we consider that the $(kk'+k+k')$ single-precision integers are pre-computed and accessible, this operation requires $4n(kk'+k+k')$ MMs and $4n(k+1)$ FMs. After basis extension, the polynomials are then represented with $k+k'$ residue polynomials over in the unified RNS basis $\mathcal{B} \cup \mathcal{B}'$.

The basis extension is followed by the tensor product over R. A naive approach for computing the tensor product requires up to four multiplications over R (Figure 5.1a). With a Karatsuba-like approach, the number of these polynomial multiplications could be reduced to 3, but at the cost of 3 more polynomial additions (Figure 5.1b). Thus, taking into account the RNS representation, there are $4(k+k')$ RPMs and $(k+k')$ RPAs for the naive approach. For the Karatsuba-like approach it represents $3(k+k')$ RPMs and $4(k+k')$ RPAs. As the polynomial multiplications are far more expensive that polynomial additions, the karatsuba-like approach is considered here. As presented in subsection 2.3.2 the implementation size of NTT is considered a power of two, even for the padded-convolution approach. The number of pre-computed values is then the same for both padded-convolution and negative wrapped convolution. Namely, $2n+1$ for each RPM and thus $(2n+1)(k+k')$ in total.



(a) Naive

(b) Karatsuba-like

Figure 5.1: Ciphertext tensor product

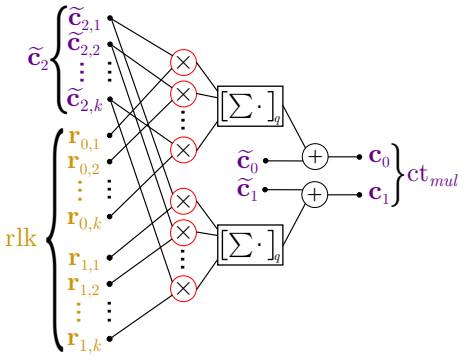


Figure 5.2: Ciphertext relinearization.

The tensor product is followed by the scale-and-round operations. As introduced at the end of Chapter 2, this operation is done in two steps. First by computing the scale-and-round

operation over the basis $\mathcal{B}'$. Second by changing the basis from $\mathcal{B}'$ to $\mathcal{B}$. The operations requires some pre-computed values: $(2k'(k+1)+k)$ single-precision integers and k floating-points in total. It is then performed in $3n(2k'(k+1)+k)$ MMs and $3n(k'+k+2)$ FMs. The resulting polynomials are back in R_q, which means that their RNS representations is only according to the basis $\mathcal{B}$.

Finally, the last operation is the relinearization that requires the knowledge of the relinearization key. This key is composed of $2k$ polynomials over R_q, which is equivalent to $2k^2$ polynomials over the R_{q_i}'s in RNS representation. As seen in Figure 5.2, the relinearization performs the scalar products of $(\mathbf{r}_{j,i})_{i=1}^{k} \in R_q^k$ with $(\tilde{\mathbf{c}}_{2,i})_{i=1}^{k} \in R_{q_1} \times ... \times R_{q_k}$, for $j \in 0, 1$. This is $2k$ multiplications and $2k$ additions over R_q, equivalent to $2k^2$ RPMs and $2k^2$ RPAs in RNS.

Table 5.1 gives the number of basic operations required to perform the basis extension (Bext) and the scale-and-round (Sc&Rnd) steps for different parameter sets. The total size of the pre-computed values is also given. The reciprocal required for efficient modular reduction is counted in the pre-computed values. These pre-computed values represent a really small storage compared to the size of the payloads to handle. And regarding the advantage of having them pre-computed, it is then considered that these values are permanently stored.

Table 5.1: Summary of number of basic operations to performs for the basis extension and the scale-and-round operations. The MM columns refer to modular multiplications and the FM columns to floating-point multiplications. Modular and floating-point additions are not considered. The column *pre.* refers to the total of pre-computed values.

Parameters				pre.	Bext. ($\times 10^3$)		Sc&Rnd ($\times 10^3$)	
n	$\log_2 q$	k	k'	kB	MM	FM	MM	FM
2^{11}	54	1	2	0.17	41	16	55	31
2^{12}	108	2	3	0.34	180	49	246	86
2^{13}	216	4	5	0.84	950	164	1,327	270
2^{14}	432	8	9	2.41	5,833	590	8,356	934
2^{15}	594	11	12	4.09	20,316	1,573	29,393	2,458
2^{16}	1026	19	20	10.68	109,838	5,243	161,022	8,061
2^{17}	2052	38	39	38.65	817,364	20,447	1,211,105	31,064

Table 5.2 summarizes the number of residue polynomial operations to perform for the ciphertext tensor product and the ciphertext relinearization. The size of the relinearization key and the total amount of data that twiddle factors represent for performing RPMs are also given. The reciprocal required for efficient modular reduction is counted in the pre-computed values. It is noticeable that the size of the relinearization key become problematic for very large parameter sets. Similarly, but at lower scale, the number of pre-computed values involved in the calculation of the RPMs grows significantly with the parameter sets.

Concluding remarks. The basis extension and scale-and-round operations are computationally dependent on the capability of performing a large amount of modular and floating-point operations. From several tenth of thousands operations for the smallest parameters up to several hundred of millions for the largest ones. The friendly parallelism over n represents

Table 5.2: Summary of number of residue polynomial operations to performs for the different tensor product and the relinearization. The size of the relinearization key is given in column "rlk" and the size for the pre-computed values in column "pre.".

Parameters				rlk	pre.	Ten. Prod.		Relin.	
n	$\log_2 q$	k	k'	kB	kB	RPM	RPA	RPM	RPA
2^{11}	54	1	2	29	86	9	12	2	2
2^{12}	108	2	3	229	287	15	20	8	8
2^{13}	216	4	5	1,835	1,032	27	36	32	32
2^{14}	432	8	9	14,680	3,900	51	68	128	128
2^{15}	594	11	12	55,509	10,552	69	92	242	242
2^{16}	1026	19	20	331,219	35,783	117	156	722	722
2^{17}	2052	38	39	2,649,750	141,297	231	308	2,888	2,888

from 2^{11} up to 2^{17} parallel and independent threads. This could be efficiently exploited by a generic SIMD architecture like a GPU. Furthermore, the pre-computed values for the computation of these two steps represent a negligible amount of permanently used GPU memory (39kB for the largest parameters).

The tensor product and the relinearization require efficient polynomial arithmetic. In a first approximation, hardware cost and execution time of RPA operations is considered negligibles. Even if the number of RPM operations to perform is rather small, its complexity is whats bound the FV.Mul&Relin efficiency in practice. Our choice is to accelerate these RPMs on dedicated hardware by exploiting our basic blocks presented in Chapters 3 and 4. Doing so, the problematic of handling the pre-computed values is solved by generating and using them on-the-fly. The relinearization key size does not allow us to directly store it locally to the hardware accelerator, thus it is considered as an operand similarly as a ciphertext. For prototyping, the dedicated hardware is considered to be implemented on a FPGA.

5.1.2 Study of the communication requirements

The flow of operations is partitioned over the FPGA and the GPU units. Figure 5.3 presents this partition, plus the quantity of data to exchange between the different steps.

A potential performance limitation already identifiable is the ping-pong communication between the FPGA and the GPU. A solution to this problem could be to mask the communications with computations (stream CUDA for GPU, DMAs and sufficient IO buffers on FPGA, ...). In this thesis, we consider the problematic of efficient computation to be more crucial than the potential communication congestion brought by this hybrid computing approach. Nevertheless, Table 5.3 gives an insight on the data loads involved for some FV's parameter sets close to those of Halevi et al. [5].

For the smallest parameterization, the unitary communications between the different computing unit are at least of 30 kB and goes up to 200 kB. Considering a flow of communication and computation for continuous ciphertext multiplications, the total amount of data to be simultaneously exchanged is roughly 500 kB. For the largest parameters, the unitary communications exchange from 79.7 MB up to 323 MB of data. This represent 923.8 MB of data to be simultaneously exchanged in the case of a continuous flow of FV.Mul&Relin operations.

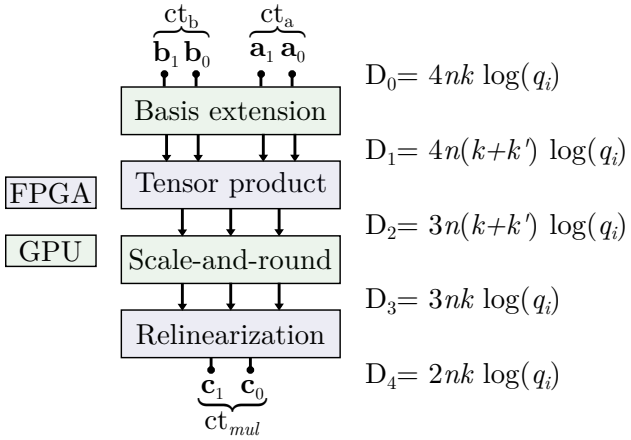


Figure 5.3: Partition of the FV.Mul&Relin operation over the GPU and FPGA. The quantity of data exchanged (in bits) is presented in between each step of the ciphertext multiplication.

Table 5.3: Quantity of data (MB) to be exchanged between the different steps of the FV.Mul&Relin calculation. The last columns group together the sum of the input and output data of the FPGA and GPU computing units for an FV.Mul&Relin operation. The size of the RNS basis elements is $\log_2 q_i = 54$-bit.

Parameters							FPGA		GPU	
n	$\log_2 q$	D_0	D_1	D_2	D_3	D_4	in	out	in	out
2^{11}	54	0.07	0.2	0.15	0.05	0.03	0.25	0.18	0.21	0.25
2^{12}	108	0.26	0.66	0.49	0.2	0.13	0.85	0.62	0.75	0.85
2^{13}	216	1.05	2.36	1.77	0.79	0.52	3.15	2.29	2.82	3.15
2^{14}	432	4.19	8.91	6.68	3.15	2.1	12.06	8.78	10.88	12.06
2^{15}	594	11.53	24.12	18.09	8.65	5.77	32.77	23.86	29.62	32.77
2^{16}	1026	39.85	81.79	61.34	29.88	19.92	111.67	81.26	101.19	111.67
2^{17}	2052	159.38	322.96	242.22	119.54	79.69	442.5	321.91	401.6	442.5

This high-level study motivates the choice of a PCIe interconnection between the different computing units. Even if the proper bandwidth requirements could only be expressed with the consideration of the computation timings, the amount of data is already an indication of the interconnection performance required. The choice of PCIe simplifies also the realization of an acceleration prototype as it is a standard connection available on most of GPU cards and FPGA prototyping boards. Table 5.4 gives a quick summary of the maximal throughput that PCIe inteconnection could achieve. This throughput is dependent on the generation and the number of lane of the instantiated PCIe interconnection.

Now that the computation workload and the communications requirements have been expressed, the next subsection propose such a hybrid computing system. FPGA for the computation of the different steps of the FV.Mul&Relin primitive.

Table 5.4: Summary of available bandwidth with the different generations of PCIe.

Gen.	year intro.	line/code	Transfer rate (GT/s)/lane	Throughput (GB/s)				
				×1	×2	×4	×8	×16
1.0	2003	8b/10b	2.5	0.25	0.5	1.0	2.0	4.0
2.0	2007	8b/10b	5.0	0.5	1.0	2.0	4.0	8.0
3.0	2010	128b/130b	8.0	0.99	1.97	3.94	7.88	15.8
4.0	2017	128b/130b	16.0	1.97	3.94	7.88	17.75	31.5
5.0	(2019)	128b/130b	32.0	3.938	7.88	15.75	31.51	63.0

5.1.3 Hybrid system overview

In this subsection, we give an overview of a hybrid computing system for the acceleration of encrypted-computing with FV.

Figure 5.4 presents our proposed hybrid computing architecture articulated around a PCIe interconnections. The control of the overall computation is made by an host CPU. The CPU is also responsible of the non-bottleneck operations required during an FV homomorphic evaluation (FV.Add and FV.Encrypt).

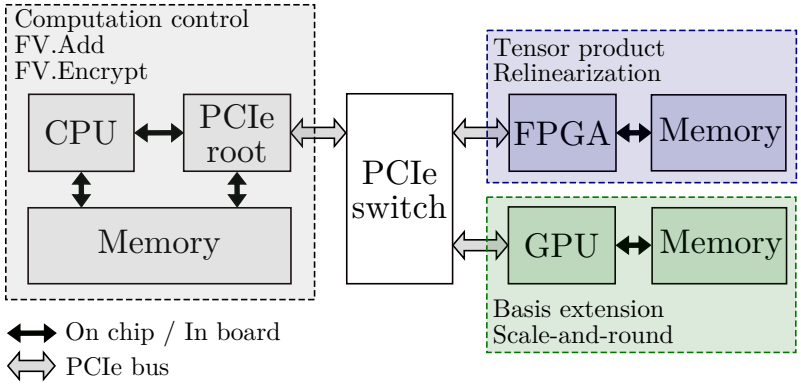


Figure 5.4: Proposed system architecture to accelerate FV homomorphic evaluation. The CPU controls the overall computations, the GPU accelerates the RNS specific functions and the FPGA accelerates the tensor products over R and the relinearization primitive.

The computation workload for the GPU is quite well identified thanks to the work of Halevi et al. [5]. During this thesis, an internship has been proposed to get a first idea of the acceleration potential bring by the GPU. The result of this work are summarized in section 5.2. The perspectives of this GPU acceleration are also discussed in this section.

The computation workload on the FPGA is expressed in terms of residue polynomial arithmetic. Namely, Residue Polynomial Multiplication (RPM) and Residue Polynomial Addition (RPA) over the residue polynomial rings (R_{q_i}'s) are required. Due to time constraint, this thesis only focuses on the definition of efficient RPM operations on FPGA. This is motivated by their predominant part in the performance bottleneck of the FV homomorphic evaluation. In particular, we choose to investigate RPM through Negative Wrapped Convolution as it gives the polynomial reduction for free. Our investigation and the perspectives of RPM through NWC are presented in section 5.3.

5.2 GPU acceleration of RNS specific functions

In this section we present preliminary results of the acceleration given by the GPU. This acceleration is highlighted by comparing an hand-made CPU and GPU implementations. Both basis extension and scale-and-round operations are implemented following Halevi et al.'s paper [5].

This implementation has been obtained with the help of Aurore Mattio who made her master thesis on the subject. This section is then a part of her internship's results that have been adapted to the current discussion.

In a first subsection a detailed description of the implemented algorithm is given. In a second subsection, the experimentation that highlights the GPU acceleration is described. Finally the results of the experimentation and some discussions conclude this section.

5.2.1 The implemented algorithms

In Section 2.4, the basis extension and the scale-and-round operations have been presented. In this subsection, the concrete algorithms implemented during Mattio's internship are detailed.

For the following, we remind the notations used in Chapter 2. The first RNS basis is composed of the element noted q_i, namely $\mathcal{B} = \{q_i\}_{i=1}^{k}$. We note $q_i^* = q/q_i \in \mathbb{Z}$ and $\tilde{q_i} = (q_i^*)^{-1} \in \mathbb{Z}_{q_i}$. The second RNS basis is composed of the element noted p_j, namely $\mathcal{B}' = \{p_j\}_{j=1}^{k'}$. And similarly $p_j^* = p/p_j \in \mathbb{Z}$ and $\tilde{p_j} = (p_j^*)^{-1} \in \mathbb{Z}_{p_j}$. Finally, the special elements associated to the unified basis $\mathcal{B} \cup \mathcal{B}'$ are: $Q = qp$, $Q_i^* = Q/q_i = q_i^* p$ and $\tilde{Q}_i = [(Q_i^*)^{-1}]_{q_i}$ for each $q_i \in \mathcal{B}$, and $Q_j'^* = Q/p_j = qp_j^*$ and $\tilde{Q'}_j = [(Q_j'^*)^{-1}]_{p_j}$ for each $p_j \in \mathcal{B}'$.

The other elements required for basis extension and scale-and-round operations are:

- $\mu_{i,j} = [q_i^*]_{p_j} \quad \forall\ (i,j) \in [1,k] \times [1,k']$.
- $\nu_{j,i} = [p_j^*]_{q_i} \quad \forall\ (j,i) \in [1,k'] \times [1,k]$.
- $\phi_j = [q]_{p_j} \quad \forall\ j \in [1,k']$.
- $\psi_j = [p]_{q_i} \quad \forall\ i \in [1,k]$.
- $\Omega_{i,j} = [\Omega_i]_{p_j}$, with Ω_i being the integer part of $\frac{\tilde{Q}_i pt}{q_i}$, $\forall\ (i,j) \in [1,k] \times [1,k']$.
- Θ_i is the fractional part of $\frac{\tilde{Q}_i pt}{q_i}$, $\forall\ i \in [1,k]$.
- $\Lambda_j = \left[\tilde{Q'}_j p_j^* t\right]_{p_j} \quad \forall\ j \in [1,k']$.

Algorithm 4 presents the implemented basis extension for each coefficient of the inputted polynomials. Similarly, the scale-and-round operation follows Algorithm 5 for each coefficient of the inputted polynomials.

In both algorithms, the calculation of the v and v' values are performed with floating-point arithmetic. The modular multiplications are implemented following the modular reduction algorithm of the NFLlib [95].

As previously stated, the straightforward parallelism to exploit with GPU is the one brought by the degree n of the polynomials. Nevertheless, as seen in Algorithm 4 and Algorithm 5, large basis size results in large inner-loops. A second level of parallelism is then potentially accessible. The implementation of this additional level of parallelism is more tricky than the first one as it involves some data dependencies between threads. This second level of parallelism is not taken into account in the following experimentation.

Now that we have basically presented how the operations are implemented, the experimentation that highlight the GPU acceleration of these operations is described.

Algorithm 4 Basis extension algorithm, following Halevi et al. [5]

Precomp: $(\tilde{q}_i, \mu_{i,j}, \phi_j)\ \forall\ (i,j) \in [1,k] \times [1,k']$
Input: $x = \{x_i\}_{i=1}^{k} \in \mathcal{B}$, coefficient of a polynomial in R_q.
Output: $X = \{X_i\}_{i=1}^{k+k'} \in \mathcal{B} \cup \mathcal{B}'$, coefficient of a polynomial in R_{qp}.

$v = 0$
for $1 \leq i < k$ **do**
    $X_i = x_i$
    $y_i = x_i \tilde{q}_i \bmod q_i$
    $v = v + \frac{y_i}{q_i}$
end for
$v = \text{round}(v)$
for $1 \leq j < k'$ **do**
    $X_{k+j} = 0$
    for $1 \leq i < k$ **do**
        $X_{k+j} = (X_{k+j} + y_i \mu_{i,j}) \bmod p_j$
    end for
    $X_{k+j} = (X_{k+j} - v\phi_j) \bmod p_j$
end for

5.2.2 Implementations, comparisons and perspectives

The purpose of our experimentation is double. First, look for an insight on the potential acceleration brought by the GPU on the RNS specific functions. Second, quantify the cost of the communications between the host and the GPU. This second information is interesting in order to have a better estimation of the criticality of the communication problematic.

Experimental setup. The basis implementation is a single threaded C/C++ version of the algorithms, running on an Intel(R) Core(TM) i7-3770 CPU at 3.40GHz, with 8GB of memory. The operating system is Red-Hat 7.0, and the code was compiled with gcc 4.8.5 with the optimization option -O2.

The GPU implementation use CUDA-8.0 and is parallelized only over the dimension n (degree of the polynomials). The host cpu is an Intel(R) Xeon(R) CPU E5-2643 v4 running at 3.40GHz, with 128GB of memory. The GPU on which are performed the operations is a Maxwell GTX Titan X clocked at 1GHz, with 12GB of memory, and connected to the host through a PCIe bridge gen.2 x16 lanes. The CUDA toolsuit version 8.0 is used to compile the code.

For each operations (basis extension and scale-and-round), the timings for input and output communications and the actual GPU computation are measured. For each set of parameters the operations have been run 1000 times to average out the timings.

Experimental results. The timing results are presented in Table 5.5. Those timings take into account the communications between the host and the GPU. The results highlight the acceleration potential of the GPU, even if the exploited parallelism is only with respect to the parameter n. For the first parameter sets, the acceleration increases with the degree n. Nevertheless, this trend is stabilizing, and even reversing for the largest ones. This could be explained both by the degree n being larger that the number of physical threads available on

Algorithm 5 Algorithm of scale-and-round for FV.Mul&Relin, following Halevi et al. [5]

Precomp1: $(\Omega_{i,j}, \Theta_i, \Lambda_j)\ \forall\ (i,j) \in [1,k] \times [1,k']$
Precomp2: $(\tilde{p}_j, \nu_{j,i}, \psi_i)\ \forall\ (i,j) \in [1,k] \times [1,k']$

Input: $X = \{X_i\}_{i=1}^{k+k'} \in \mathcal{B} \cup \mathcal{B}'$, coefficient of a polynomial in R_{qp}.
Output: $x = \{x_i\}_{i=1}^{k} \in \mathcal{B}$, coefficient of a polynomial in R_q such that $x = [\lfloor t/q \cdot X \rceil]_q$.

$v' = 0$
for $1 \leq i < k$ **do**
    $v' = v' + X_i\Theta_i$
end for
$v' = \text{round}(v')$
for $1 \leq j < k'$ **do**
    $\text{tmp} = X_{k+j}\Lambda_j \bmod p_j$
    for $1 \leq i < k$ **do**
        $\text{tmp} = (\text{tmp} + \Omega_{i,j}X_i) \bmod p_j$
    end for
    $x'_j = (\text{tmp} + v') \bmod p_j$
end for ▷ At this point, $x' = \{x'_j\}_{j=1}^{k'} = [\lfloor t/q \cdot X \rceil]_p$ in basis $\mathcal{B}'$
$v = 0$ ▷ Hence, we change the basis: $\mathcal{B}' \to \mathcal{B}$
for $1 \leq j < k'$ **do**
    $y_j = x'_j\tilde{p}_j \bmod p_j$
    $z_j = \frac{y_j}{p_j}$
    $v = v + z_j$
end for
$v = \text{round}(v)$
for $1 \leq i < k$ **do**
    $x_i = 0$
    for $1 \leq j < k'$ **do**
        $x_i = (x_i + y_j\nu_{j,i}) \bmod q_i$
    end for
    $x_i = (x_i - v\psi_i) \bmod q_i$
end for ▷ We only keep the residue in basis $\mathcal{B}$, namely $\{x_i\}_{i=1}^{k}$

Table 5.5: Timing in milliseconds for the basis extension and the scale-and-round operations on CPU and GPU, for different parameter sets.

Parameters		CPU		GPU			
n	$\log_2 q$	Bext	Sc&Rnd	Bext	su	Sc&Rnd	su
2^{11}	54	0.585	0.977	0.079	$\times$**7.4**	0.064	$\times$**15.3**
2^{12}	108	1.688	2.816	0.162	$\times$**10.4**	0.135	$\times$**20.9**
2^{13}	216	7.724	13.411	0.524	$\times$**14.7**	0.409	$\times$**32.8**
2^{14}	432	46.103	80.274	1.604	$\times$**28.7**	1.583	$\times$**50.7**
2^{15}	594	162.163	286.106	4.81	$\times$**33.7**	4.634	$\times$**61.7**
2^{16}	1026	875.98	1,532.584	29.565	$\times$**29.6**	29.856	$\times$**51.3**
2^{17}	2052	6,836.377	11,498.993	235.866	$\times$**29**	275.89	$\times$**41.7**

Table 5.6: Timing details in microseconds for the GPU acceleration of the basis extension and the scale-and-round operations. The in/out columns presents the timing for loading and unloading data from the GPU.

Parameters		Bext			Sc&Rnd			BW (GB/s)	
n	$\log_2 q$	in	cmpt	out	in	cmpt	out	in	out
2^{11}	54	9	22	47	15	31	17	2.55	1
2^{12}	108	18	39	104	26	61	47	4.97	1.48
2^{13}	216	44	109	369	67	179	162	7.38	1.61
2^{14}	432	300	432	871	594	477	512	3.62	2.3
2^{15}	594	662	2,855	1,293	1,565	2,629	440	4.1	5.61
2^{16}	1026	2,175	20,619	6,770	3,759	22,736	3,360	5.01	2.99
2^{17}	2052	5,793	205,891	24,181	11,740	252,360	11,789	6.88	3.36

the GPU, and by the size of the RNS basis that makes the inner-loops more costly. Another interesting point is that for all the parameter sets except for the two largest ones, the basis extension is slower than the scale-and-round operation. Indeed, it should not be the case as the second operation is more complex than the first. This motivates the investigation of communication costs.

The details of the GPU timing is given in Table 5.6. For the four first parameter sets, the time spent in communications is more than twice as long as the actual computations. Even for the larger set, the timing spent in communications is roughly 13% (respectively 9%) of the overall timing for basis extension (respectively scale-and-round). The impact of communications is mainly due to unloading the results from the GPU. And comparing the mean bandwidth usage for the input and output communications, it seems that the input/output communication capability of the GPU is unbalanced.

Considering now the communication payloads on input and output of each operations. The basis extension takes a polynomial in $\mathcal{B}$ and output a polynomial in $\mathcal{B} \cup \mathcal{B}'$, the data amount on output is twice the amount on input. The scale-and-round takes a polynomial in $\mathcal{B} \cup \mathcal{B}'$ and outputs a polynomial in $\mathcal{B}$, the data amount on output is twice less the amount

on input. Hence, the strange timing of Table 5.5 seems to be explained by the unbalanced communication capability of the GPU.

Due to time constraints, we did not investigate further this phenomenon. But this is an important information for one who wants to instantiate the proposed hybrid computing system. This indicates that the communication with the GPU should be more finely addressed for a concrete implementation. Nevertheless, even with salient communication timing, the operations benefit of a non-negligible acceleration.

5.3 Exploration of efficient RPM designs

Our proposal for a hybrid computing architecture delegates residue polynomial arithmetic to dedicated hardware. According to the recent profiling of Halevi et al. [5], the most critical operations to implement in FPGA are the Residue Polynomial Multiplications (RPM).

This section presents an example of efficient NTT-based residue polynomial multiplications using our basic blocks presented in previous chapters. In particular it studies the acceleration of RPM through data-flow oriented Negative Wrapped Convolution (NWC), with on the fly generation of twiddle factors.

Our proposal for a NWC-based residue polynomial multiplier is detailed in the first subsection. A proof-of-concept implementation of this RPM design is presented in a second subsection. It includes also a projection of this design on the wide range of parameter sets. Finally, a third subsection concludes by highlighting the main conclusions of this RPM approach and drawing some perspectives.

5.3.1 Reminder of previous chapters

Chapter 3 has presented a solution to generate multi-field NTT circuits. As NTTs are similar to a Discrete Fourier Transform (DFT), our work has explored the generalization of the hardware backend of the SPIRAL tool, from Milder et al. [97], to generate NTT designs in addition to DFT designs. Hence, we have modified the DFT hardware produced by SPIRAL to convert it into a practical NTT structure for RPMs by making two sets of changes. First, we have replaced the DFT's arithmetic blocks with those that perform modular arithmetic. Second, we have adapted the twiddle factors handling.

We remind that due to time constraints, we focused our contribution on data-flow NTT circuits. For the following discussion, we consider these data-flow multi-field NTT circuits as basic blocks. Nevertheless, it is important to consider that these multi-field NTTs dissociate the twiddle path from the actual data path of the NTT. The data path implements the NTT algorithm itself, and the twiddle path instantiates the circularly-buffered handling of the twiddle factor sets. Consequently, an NTT twiddle path may feed multiple parallel NTT data paths.

Chapter 4 has presented our solution to avoid extra costs on communication and/or memory on the accelerator for managing the twiddle factor sets. That is, a twiddle factor set generator that generates the pre-calculated values for NTT-based RPMs, without major impact on hardware performance and cost.

We remind that a twiddle factor set is a sequence of powers of a special element in the considered finite-field. Namely, if we note ψ_i this special element over the finite-field $\mathbb{Z}_{q_i}$,

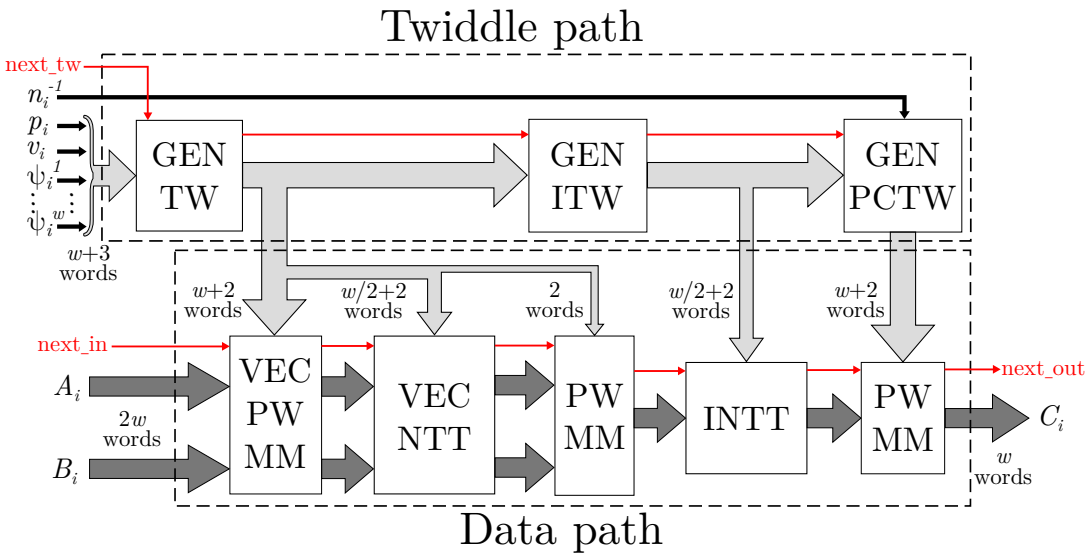


Figure 5.5: Residue Polynomial Multiplier (RPM) flow.

the twiddle factor set is the sequence $\Psi_i = \{\psi_i^j\}_{0 \leq j < n}$. By considering data-flow oriented RPMs, the impact on computational performance is avoided by adapting the throughput of the twiddle factor set generator with the throughput of the RPM data path.

5.3.2 Hardware design of an RPM through NWC

As we have seen in Chapter 2, both Padded Convolution (PC) and Negative Wrapped Convolution (NWC) are suitable for implementing RPM in our context. In this subsection we explored an RPM design through NWC as it does not require the implementation of a polynomial modular reduction. Nevertheless, it is considered that the two approaches are quite comparable in terms of hardware cost and performances, assuming that efficient data-flow oriented polynomial modular reduction is possible. But, the veracity of this assumption has not been verified in this thesis.

In the following description, multiplication refers to multiplication over the ring $\mathbb{Z}_{p_i}$. As presented in section 3.2.3, modular multiplications are performed following the NFLlib algorithm [95] for modular reduction. This requires the prime q_i and the reciprocal v_i as inputs.

Moreover, we remind that an NWC requires the knowledge of a n-th primitive root of -1 over the considered finite-field $\mathbb{Z}_{q_i}$. In the following, we note ψ_i such a n-th primitive root of -1. This means that $\psi_i^n = q_i - 1 \mod q_i$ and for all $k < n$ $\psi_i^k \neq q_i - 1 \mod q_i$. With $\Psi_i = (\psi_i^k)_{k=0}^{n-1}$ and $\Psi_i^{-1} = (\psi_i^{-k})_{k=0}^{n-1}$, the NWC over $\mathbb{Z}_{q_i}$ is performed by computing:

$$\mathbf{c}_i = \Psi_i^{-1} \odot \mathrm{INTT}_{n,i}\left(\mathrm{NTT}_{n,i}(\Psi_i \odot \mathbf{a}_i) \odot \mathrm{NTT}_{n,i}(\Psi_i \odot \mathbf{b}_i)\right). \tag{5.1}$$

The overall architecture flow is presented in Figure 5.5 without the artificial latency for representation simplicity. The architecture is generic regarding the size of the NTT n, the width of the data path w (called streaming-width), and the prime size s in bits, with n and w being powers of two and $s \leq 64$.

There are two parallel paths in this architecture: the twiddle path and the data path. On one side, the twiddle path feeds the data path with the appropriate twiddle values, consistent with the actual polynomial ring ($R_{q_i} = \mathbb{Z}_{q_i}[X]/(X^n + 1)$) of the residue polynomials A_i and B_i. On the other side, the data path performs the negative wrapped convolution of the two input polynomials seen as n-sequence of coefficients.

Data path. Five distinct steps are required to perform a NWC on inputted polynomials.

The first step is performed by VEC PW MM and consists of inner-products of the input polynomials with the weight-vector $\Psi_i = (\psi_i^j)_{0 \leq j < n}$ to output the polynomials $\Psi_i \odot A_i$ and $\Psi_i \odot B_i$. Only the n first elements of the twiddle factor set are required.

The second step VEC NTT computes forward NTT on each input and outputs simultaneously the transformed polynomials $NTT_i(\Psi_i \odot A_i)$ and $NTT_i(\Psi_i \odot B_i)$. It needs $\Omega_i = \{\omega_i^j\}_{0 \leq j < n/2} = \{\psi_i^{2j}\}_{0 \leq j < n/2}$ which is a subset of the values involved in Ψ_i.

The third step PW MM corresponds to the inner-product of the two weighted polynomials in the NTT domain $NTT_i(\Psi_i \odot A_i) \odot NTT_i(\Psi_i \odot B_i)$. Only the pair (q_i, v_i) is required to perform this inner-product.

The fourth step INTT reverts the polynomial from the NTT domain, and twiddles $\Omega_i^{-1} = \{\omega_i^{-j}\}_{0 \leq j < n/2} = \{\psi_i^{-2j}\}_{0 \leq j < n/2}$ are required. Ω_i^{-1} is a subset of the weight-vector Ψ_i^{-1} used in the fifth step.

Finally, the fifth step PW MM performs, in a single step, the scaling by $n_i^{-1} \bmod q_i$ required at the end of INTT, and the inner-product with the weight-vector $\Psi_i^{-1} = (\psi_i^{-j})_{0 \leq j < n}$. Thus, only the n last elements of the twiddle factor set are required.

Twiddle path. As emphasized in the description of the data path, the twiddle values are not all required at the same time. Consequently, the computation of the twiddles is decomposed in three steps.

The first step consists of the generation of the n first powers of ψ_i, namely $\Psi_i = \{\psi_i^j\}_{0 \leq j < n}$. Along with the corresponding (q_i, v_i) pair, they feed the first three steps of the data path. The twiddle generator GEN TW, described in Chapter 4, only requires the first w elements $(\psi_i^1, ..., \psi_i^w)$ of the Ψ_i sequence. It then outputs the n sized sequence at a rate of w elements per cycle, after a certain latency.

The second step GEN ITW outputs, after a certain latency, the sequence $\Psi_i^{-1} = \{\psi_i^{-j}\}_{0 \leq j < n}$ at a rate of w elements per cycle. The computation of this sequence is done by first computing the sequence $\{\psi_i^{-(n-j)}\}_{0 \leq j < n}$, and then reordering it to obtain $\{\psi_i^{-j}\}_{0 \leq j < n}$. The sequence to reorder is computed by subtracting each element of Ψ_i from p_i[I]. Half of the Ψ_i^{-1} sequence feeds the inverse NTT, because only $\{\psi_i^{-2j}\}_{0 \leq j < n/2}$ is required.

The third step GEN PCTW scales the sequence outputted by GEN ITW by n_i^{-1} (inverse of n in $\mathbb{Z}_{q_i}$). It then feeds the point-wise multiplier (again with (q_i, v_i)) at the end of the data-flow which, thus, can complete the negative wrapped convolution.

Data-flow operations. The overall architecture is data-flow oriented, meaning that it starts a new polynomial multiplication, over a different RNS channel (polynomial ring $\mathbb{Z}_{q_i}[X]/(X^n + 1)$), every $T = n/w$ cycles. From now on, T will be identified as the throughput of the RPM design.

[I] First $\psi_i^n = -1 \bmod p_i$ implies $\psi_i^{2n} = 1 \bmod p_i$. Then, $\psi_i^{-(n-j)} = \psi_i^{-n}\psi_i^j = \psi_i^n\psi_i^j \bmod p_i$. And, lastly, $\psi_i^n\psi_i^j = (q_i - 1)\psi_i^j = q_i - \psi_i^j \bmod q_i$. Hence, $q_i - \psi_i^j = \psi_i^{-(n-j)} \bmod q_i$.

For the RPM to achieve a throughput of $T = n/w$ cycles, the different twiddle sequences, computed by the twiddle path, have to be generated with the same throughput. Chapter 4 details the twiddle factor generator that is able of generating the initial sequence $\Psi_i = \{\psi_i^j\}_{0 \leq j < n}$ from the knowledge of $w << n$ seed elements $(\psi_i^1, ..., \psi_i^w)$. Then, the generation of subsequent sequences with the required throughput is quite straightforward.

5.3.3 Proof-of-concept implementation

This subsection describes the result of a proof-of-concept implementation for a set of small cryptosystem parameters $n = 4096$, $w = 2$ and $s = 30$. Then, it studies the scaling of our approach to sets of larger cryptosystem parameters by projecting the change of SPIRAL generated DFT into multi-field NTT. This part allows us to explore performances of the RPM architecture on most of the parameter sets from [5]. Finally, it shows the positive impact of the twiddle set generator on the scalability of the overall RPM for BFV-like homomorphic schemes.

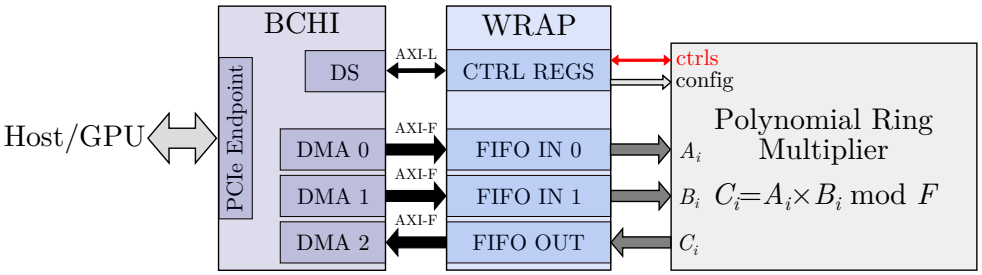


Figure 5.6: Proposal of a plausible system integration of a Residue Polynomial Multiplier.

Figure 5.6 illustrates the implemented design. The targeted prototyping board is an Alpha-Data board ADM-PCIE-7V3, embedding a Xilinx Virtex 7 xc7vx690t, and connected to host PC through PCIe Gen3 ×8 lanes. A basic AXI to IP wrapper has been instantiated with 3 FIFOs large enough to buffer 2 residue polynomial each. Namely, three fifo of depth 1024×256-bit. In addition to the basic status signals, the wrapper's control registers store two configuration sets, which consist in the twiddle seeds along with the prime and associated reciprocal $(q_i, v_i, \psi_i^1, ..., \psi_i^w)$. Namely, 2×128-bit of configuration registers.

The proof-of-concept NWC design is synthesized, placed and routed along with the Bridge Host Controler Interface (BHCI) IP, provided by Alpha-Data, controlling the PCIe and DMAs that access the RPM design. Synthesis, placement and route have been completed with integrated tools of Xilinx Vivado 2016.3. The resulting implementation achieved a running frequency of 200MHz without particular effort.

In Table 5.7, the resource utilization post-implementation is shown for the proof-of-concept NWC design. Considering only the NWC design w.r.t. the FPGA resources, the critical resources are DSP and BRAM tiles with respectively 14,4% and 14,2% utilization, 12,5% for LUT, and 8,3% for LUTRAM. The larger part of the resource utilization comes from the NTTs (70,2% of DSPs, 70,8% of BRAMs, and 77,4% of LUTs). The twiddle path, embedding our twiddle factor generator, uses roughly around 10%-13% of DSPs, BRAMs and LUTs. The inner-products in the overall data-flow use 17% of DSPs, and the various latencies synchronizing the data path and the twiddle path together represent 20% of the BRAMs utilization. The hardware cost for the synchronization (*Others* column) is considered constant as it becomes relatively small for larger n.

Table 5.7: Resource utilization post implementation on a Virtex xc7vx690t. Synthesis, placement and route using Xilinx Vivado 2016.3. Frequency 200MHz.

Ressources		**RPM**						BCHI
type	available	%	**total**	NTT	MM	GTW	Others	& WRAP
LUT	432,368	**12.5**	**54,188**	41,964	5,198	5,906	1,120	27,775
LUTRAM	173,992	**8.3**	**14,402**	10,710	2,056	1,550	86	5,425
FF	864,736	**7.7**	**66,444**	50,961	6,755	7,761	967	39,614
BRAM	1,470	**14.2**	**208**	147	0	21	40	153
DSP	3,600	**14.4**	**517**	363	88	66	0	48
IO	600	–	–	–	–	–	–	59
Pcie	3	–	–	–	–	–	–	1

The concrete test of the implemented design has not been completed due to some wrapper issues. Indeed, our wrapper has been implemented with AXI-stream port for the connection with the BCHI. Unfortunately, we didn't succeed to make the DMAs to work with AXI-stream; we only managed to make them work with AXI-full connections. Hence, we should have modified our IP wrapper in consequence, but due to time constraints, this have not been possible yet.

This proof-of-concept has nevertheless validated the approach from a functional point-of-view. This also allowed us to obtain the relative cost of the system integration for this approach. And finally, it also gave us a theoretical running frequency from which we can project some performances and bandwidth requirements for larger parameter sets.

5.3.4 Projections over FV's parameter sets

In order to analyze the scalability of our hardware acceleration approach, its characteristics under the influence of concrete parameter sets are studied in this section. What is interesting us at this point is an estimation of the achievable performance of a data-flow oriented NTT-based RPM. These performance are relevant if and only if it is practically possible to achieve the design. Hence, this section presents both projection on hardware costs and on the computing performance of our NWC design.

Projection methodology. The following estimations are built on two basis: the previously presented implementation for $n = 4096$, $w = 2$ and $s = 30$, and the estimated changes of SPIRAL generated DFT into NTT. For each estimation, we analyze the resource count of the appropriate DFT design, and adjusted the costs of the arithmetic, memories, and required bandwidth to match the requirements of the corresponding modified NTT design. When considering a data-flow design, going from DFT to NTT mainly impacts the hardware cost of the design, as the throughput does not change for a specific transform size. Similarly, the impact on the latency of changing DFT into NTT is not considered here regarding the number of pipelined RPM to perform in practice.

For the following discussions, a DSP refers to a 7 series DSP48E1, and a BRAM refers to a 36Kb Block RAM. Estimations of resource utilization are based on the corresponding Xilinx IP core generators for unsigned multiplier and single port RAM memories. Note that

no potential synthesizer optimizations have been taken into account. Finally, the number of LUT is neglected because it does not appear as a critical resource in practice.

We remind that in Chapter 3 the concrete synthesis results are given for the twiddle path of the multi-field NTT. Similarly, Chapter 4 details the concrete synthesis results for the twiddle factor set generator. In the following discussions, we do not consider the results of the synthesis, but those of our projection. These projections are sufficient to validate the consistency of our RPM design, although these projections are pessimistic. For instance, for the parameter set ($n = 2^{14}$, $w = 4$, $s = 30$), our projections estimate that one twiddle path of a multi-field NTT is utilizing 112 BRAMs slices and our synthesis result shows that it actually uses only 88 of them (see Chapter 3). Another example for the estimation of the GENTW module resources utilization, for the parameter set ($n = 2^{14}$, $w = 4$, $s = 30$) our projections estimate the usage of 48 BRAMs and 48 DSPs, and our synthesis result indicates 42 BRAMs and 44 DSPs (see Chapter 4).

Projections on hardware costs. The following discussion refers to Figure 5.7 presenting our projection results on resource utilization. All sizing parameters n, w and s have a significant impact on resource utilization. Figure 5.7a presents the influence of the degree of the polynomials n, Figure 5.7b the influences of the streaming width w, and Figure 5.7c the influence of the size of the primes. The *limit* value represents the available hardware/bandwidth resource within Alpha-Data board used for our proof-of-concept. This *limit* takes also into account the BCHI and the wrapper resource utilization plus a 10% margin for a concrete implementation.

The degree n of the handled polynomials mainly impacts the number of BRAM required. Its influence on DSP utilization is linear due to the number of stages being in $O(\log_2 n)$ and n being power of two. The bandwidth requirement to fully load the design is constant due to the data-flow approach that makes it only dependent on the streaming-width w, the prime size s, and the running frequency. The estimated number of BRAM used goes from 226 (roughly 1 MB of data) for $n = 2^{12}$ up to 1074 (roughly 4.8 MB of data) for $n = 2^{15}$. With larger degree n ($> 2^{15}$), the BRAMs utilization limits the feasibility of the design. The larger part of the utilization (> 64 %) is used to implement the permutations of the NTT data paths. Hence, a potential solution to address larger degree is to implement the design on a FPGA with increased number of available BRAMs, or to improve the resource efficiency of the NTT permutations.

The streaming width w improves the throughput of the RPM significantly. Indeed, the number of cycles between two RPM computations is $T = n/w$. For instance, considering $n = 2^{14}$, it is $32,768$ cycles for $w = 2$ down to 4096 for $w = 16$. But due to a larger streaming-width, this has an heavy drawback on the DSP utilization (from 636 up to $4,836$ DSP slices), and on the required communication bandwidth (from 4.5 GB/s up to 36 GB/s for a 200MHz clock frequency). Thus, increasing the streaming-width to improves the performances of the design seems not to be a viable solution.

The element size s (size in bit of the RNS basis elements) is interesting to consider because it influences the size of the RNS basis for a given parameter q. Doing so reduces the number of RPM to perform for a ciphertext multiplication. This parameter has a balanced impact on BRAM utilization, DSP utilization and required communication bandwidth. Nevertheless, some increments of s have a more significant impact on DSP utilization, this is due to the architectural characteristic of the DSP elements. Similarly, with increased s, the latency of

modular multiplications may be larger if one wants to keep the same running frequency.

To conclude on these projections, they show the feasibility of RPM through data-flow NWC. Nonetheless, they indicate some limitations to address very large parameter sets $n > 2^{15}$ due to BRAM utilization. They also gives some indications on how to improve the computing performance of the design. Our recommendations on this matter would be to tune the size of the RNS basis element s rather than the streaming-width w.

Even if these projections are pessimistic, they nonetheless express the flexibility of our RPM design for our application context. We assume that this is sufficient to validate the relevance of the following timing projections.

Execution performance scalability. To study the execution performance of the RPM design, we estimate the resulting timing and compare with the profiling from Halevi et al. [5] already presented in Chapter 2. Table 5.8 reminds this profiling over which execution performance comparisons are based.

Table 5.8: Timing estimate of the FV.Mul&Relin primitive derived from the profiling of Halevi et al. [5]. Single-threaded mode, Linux CentOS, Intel Core i7-3770 CPU 4 cores at 3.40GHz and 16 GB of RAM; plaintext space $t = 2$, $s = 54$-bit, security $\lambda > 128$. We remind that L is the multiplicative depth evaluation capability of the FV scheme.

L	n	S_q	k	k'	Total	CRT ext. & Scaling		**Mul.RPM**		**Relin.RPM**		Others	
					ms	ms	%	**ms**	%	**ms**	%	ms	%
1	2^{11}	54	1	2	3.6	1.3	37.2	**1.8**	50.3	**0.4**	10.9	0.1	1.6
5	2^{12}	108	2	3	12.7	3.8	30.3	**6.1**	48.4	**2.5**	19.3	0.3	2.0
10	2^{13}	216	4	5	57.6	14.4	25.0	**24**	41.6	**17.8**	30.8	1.5	2.6
20	2^{14}	432	8	9	252	71.3	28.3	**100.9**	40	**74.4**	29.5	5.7	2.2
30	2^{15}	594	11	12	887	233.1	26.3	**315.2**	35.5	**315.4**	35.6	23.3	2.6

In Table 5.9 the acceleration results over different parameter sets from [5] are presented. The distinction is made between the RPMs involved in the tensor product (Mul.RPM) and the RPMs involved in the relinearization (Relin.RPM). The number of RPM performed during these steps depends on RNS basis sizes k and k'. Namely, the tensor product requires $3(k+k')$ RPMs, and the relinearization requires $2k^2$ of them. The timings are estimated considering all the RPMs being performed in a single flow.

In the first part of the table, the estimated acceleration of our NWC-design for different FV's parameter sets is highlighted. It is noticeable that the approach gives a significant speedup over the RPM computations: at least 22.2 for the RPM of the tensor product and 4 for the RPM of relinearization. Furthermore, this speedup remains relevant with the growth of the parameter sets: from 22.2 up to 31.3 for tensor product and from 4.0 up to 9.5 for relinearization. Another element that has to be pointed out being the better speedup obtained for the tensor product than for relinearization. This element is discussed in details a little further.

The second part of the table shows the expected improvement of the acceleration when increasing the streaming-width w of our design. The speedup is estimated up to 219 for

the tensor product and up to 32.3 for the relinearization. Nonetheless, we remind that the hardware cost to do so is non-negligible, and that the required bandwidth to concretely achieve this performances is particularly high.

Finally, the third part shows the concrete impact of reducing the size RNS basis by increasing the size of their elements. Namely, the speedup lays from 27.4 up to 54.8 for the RPMs during tensor product and from 4 up to 18.5 for the RPMs during relinearization. We recall that the impact on hardware cost is relatively light. Hence, the resulting speedup improvement are quite interesting to consider for a concrete implementation.

Table 5.9: Estimated performance of our RPM design over the different parameter sets. The RPM throughput is estimated with a 200MHz clock frequency. The number of cycles between two consecutive RPM calculation is $T = n/w$. The number of RPM for the tensor product is $3(k + k')$ and $2k^2$ for relinearization. The speedup are expressed relatively to the estimated timings of Table 5.8.

Parameters							RPM	**Mul.RPM**		**Relin.RPM**	
L	n	S_q	s	k	k'	w	1/ms	#	ms(**su**)	#	ms(**su**)
1	2^{12}	54		2	3		195.3	15	0.08(×**23.4**)	8	0.04(×**9.5**)
5	2^{13}	108		4	5		97.7	27	0.28(×**22.2**)	32	0.33(×**7.5**)
10	2^{13}	216	30	8	8	2	48.8	48	0.98(×**24.4**)	128	2.62(×**6.8**)
20	2^{14}	432		15	15		24.4	93	3.69(×**27.4**)	450	18.4(×**4.0**)
30	2^{15}	594		20	21		12.2	126	10.1(×**31.3**)	882	65.5(×**4.8**)
20	2^{14}	432	30	15	15	2	24.4	93	3.69(×**27.4**)	450	18.4(×**4**)
						4	48.8		1.84(×**54.8**)		9.22(×**8.1**)
						8	97.7		0.92(×**109.5**)		4.61(×**16.1**)
						16	195.3		0.46(×**219**)		2.30(×**32.3**)
20	2^{14}	432	30	15	15	2	24.4	93	3.69(×**27.4**)	450	18.4(×**4**)
			41	11	11			69	2.70(×**37.3**)	242	9.91(×**7.5**)
			51	9	9			54	2.21(×**45.6**)	162	6.64(×**11.2**)
			58	8	8			48	1.97(×**51.3**)	128	5.24(×**14.2**)
			62	7	8			45	1.84(×**54.8**)	98	4.01(×**18.5**)

We would like to come back to the differences in acceleration between the tensor product and the relinearization. This is explained because our approach accelerates RPM operations rather than NTT operations. When considering the complexity at RPM level, some optimization possible at NTT level are not accessible. For instance, the relinearization key could be stored already in NTT domain, hence the number of NTT to perform in the relinearization is k^2+2k. This has to be compared to the $2k^2$ operations to performs at RPM level. But the other way around, considering the complexity at NTT level would make us consider more operations during the tensor product.

The number of operations to performed for the different abstraction levels are compared in Table 5.10. Our discussion here only considers the operations of similar complexity (RPM and NTT). At RPM level, we have roughly half less operations to perform than at NTT level during the tensor product, but during relinearization there is almost twice more operations to

perform when the RNS basis get larger. But it also appears that the loss during relinearization is not compensated by our gain during tensor product. Namely, for the the setting ($n = 2^{14}$, $\log_2 q = 432$), we perform 68 less operations at RPM level than NTT level during the tensor product, but 99 more operations during the relinearization. Consequently, for large parameter sets, its seems more relevant to consider the acceleration of operation at NTT level rather than at RPM level.

Table 5.10: Differences of considering the calculation complexity at RPM level rather than NTT level. The count of number of operations for the NTT level are based on the implementation of Halevi et al. [5]. The size of the RNS basis elements is $s = 54$-bit.

Parameters				Ten. Prod.			Relin.		
n	$\log_2 q$	k	k'	RPM	NTT	MM	RPM	NTT	MM
2^{11}	54	1	2	9	21	18,432	2	3	4,096
2^{12}	108	2	3	15	35	61,440	8	8	32,768
2^{13}	216	4	5	27	63	221,184	32	24	262,144
2^{14}	432	8	9	51	119	835,584	128	80	2,097,152
2^{15}	594	11	12	69	161	2,260,992	242	143	7,929,861
2^{16}	1026	19	20	117	273	7,667,711	722	399	47,316,974
2^{17}	2052	38	39	231	539	30,277,616	2,888	1,520	378,535,800

5.3.5 Concluding remarks

In the literature, the main problematic for the definition of scalable RPM designs is the handling of the multiple twiddle factor sets brought by the RNS representation. In this thesis, we have explored the on-the-fly computations of these twiddle factor sets. These specific contributions have been described in previous chapters. In this section we have shown an example of the exploitation of these basic blocks for the definition of a scalable RPM designs.

In particular, a proof-of-concept NWC-based RPM has validated the feasibility of our approach. The projections that have followed give good insurances on the flexibility brought by our basic blocks. From the best of our knowledge, our design strategy is the first that gives performances enhancement while being able to scale up to large parameter sets for an achievable hardware cost and communication bandwidth in the context of Leveld-FHE.

Moreover, some improvements are possible. The highlighted limitations of considering the complexity at RPM rather than NTT level motivates definition of residue polynomial arithmetic at a lower abstraction level. Doing so breaks the rigidity of a large data-flow RPM design. This could be replaced by several smaller and independent NTT data-flow, and hence increasing the exploited parallelism at NTT level.

In the next section, the results of the exploration of FPGA acceleration of RPMs and those of the exploration of GPU acceleration for RNS specific functions are concatenated. This allows us to conclude on the estimated acceleration of our hybrid computing system for encrypted-computing with FV.

5.4 Conclusion

In the two previous sections, both the acceleration of GPU for RNS specific functions and the acceleration of an NWC-based RPM on FPGA have been explored.

To motivate the pertinence of our hybrid computing system, we estimate the acceleration obtained for the computation of one FV.Mul&Relin. Table 5.11 presents the resulting projections when considering 54-bit primes elements to compose the RNS basis. The communication timing for GPU accesses has been taken into account.

Table 5.11: Projection of the acceleration obtained with our hybrid computing system proposal for the FV.Mul&Relin primitive. Communication timings obtained during the experimentation on GPU are included. The RNS columns regroup basis extensions and scale-and-round operations. The RPM columns regroup RPMs for tensor product and for relinearization. RPMs timings are for $w = 2$ and $s = 54$.

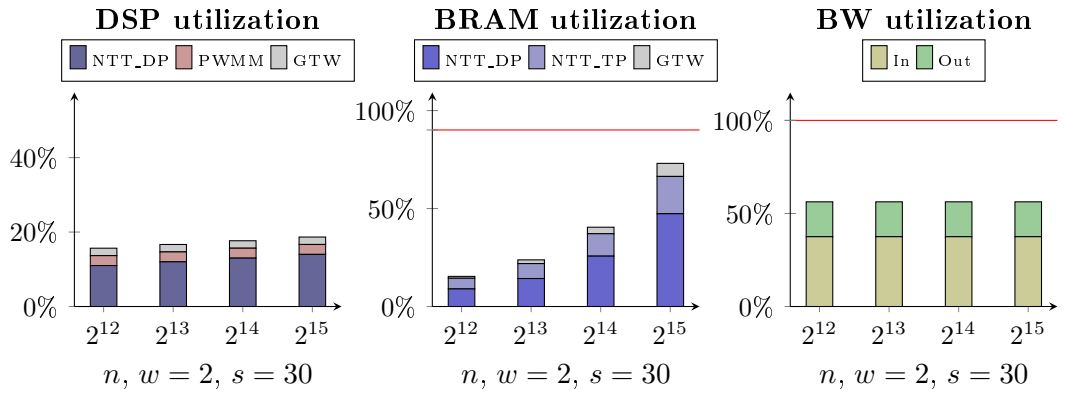
Parameters			Timing basis from [5]				Our hybrid system			
L	n	$\log_2 q$	**Total**	RNS	RPM	Misc.	**Total**	speedup	RNS	RPM
1	2^{11}	54	**3.6**	1.3	2.2	0.1	**0.67**	**×5.37**	0.51	0.06
5	2^{12}	108	**12.7**	3.8	8.6	2	**3.28**	**×3.87**	1.05	0.23
10	2^{13}	216	**57.6**	14.4	41.8	2.6	**7.13**	**×8.08**	3.32	1.21
20	2^{14}	432	**252**	71.3	175.3	2.2	**20.7**	**×12.17**	11.17	7.33
30	2^{15}	594	**887**	233.1	630.6	2.6	**61.21**	**×14.49**	33.14	25.47

Our hybrid computing approach is quite well justified by these estimations. It should be noted that the greatest acceleration concerns the parameterization for the greatest multiplicative depth in the encrypted domain. It allows us to consider a multiplication of encrypted data up to 14 times faster than the well-optimized software version of Halevi et al. [5]. This acceleration reduces the time for en encrypted multiplication to 62 ms for an evaluation capacity of a multiplicative depth of 30, and to 21 ms for a multiplicative depth of 20.

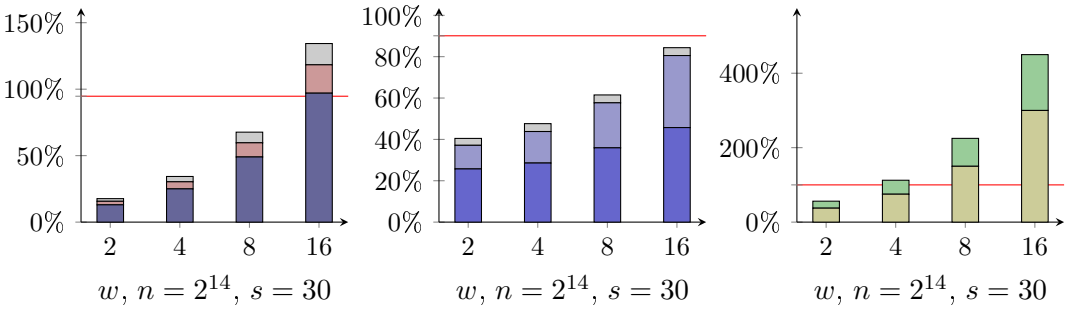
The dedicated hardware acceleration for RPMs is based on the two contributions from Chapters 3 and 4 that allow the practical feasibility of this accelerator: efficient handling of twiddle factor sets for multi-field NTT circuits, and on-the-fly generation of these sets. Together, these approaches significantly reduce the number of BRAMs used to store twiddle factors: from 37% to 84% for multiplicative depths from 5 to 30. They also reduce the required bandwidth to fill up the RPM accelerator with the twiddle factors from hundreds of MB/s to only several kB/s.

Nevertheless, a limitation of this projection is that we make the assumption that the complexity of the tensor product and relinearization is equivalent to the complexity of computing the underlying RPMs. It is not totally true as these RPMs operations are in practice interleaved with Residue Polynomial Additions (RPA). Hence, future works should propose an FPGA acceleration for both RPMs and RPAs. It is assumed that our contributions on the definition of multi-field NTT circuits and twiddle factor generators could definitely help this purpose.

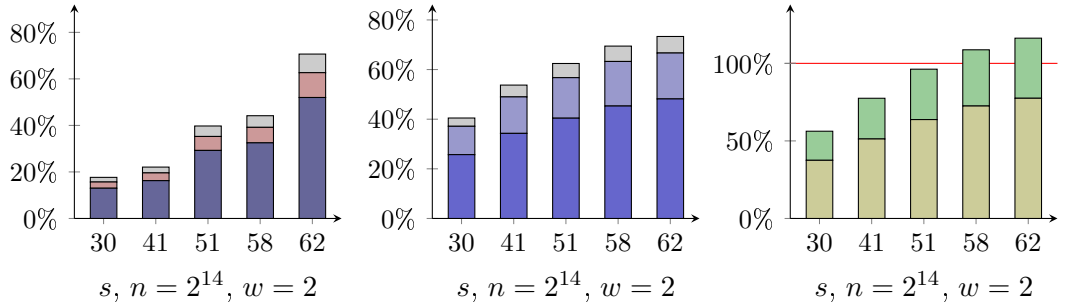
These prospective results motivate the refinement of the dedicated accelerator for polynomial arithmetic as well as the realization of the prototype of hybrid acceleration.



(a) Influence of n over utilization of FPGA resources.



(b) Influence of w over utilization of FPGA resources.



(c) Influence of s over utilization of FPGA resources.

Figure 5.7: Estimation of resource utilization under the influence of sizing parameters. The relative resource utilization is estimated with respect to the quantity of resources of a virtex 7 xc7vx690t for DSP and BRAM, and with respect to the bandwidth capacity of a PCIe gen 3 x8 lanes for the required bandwidth. Bandwidth requirements are expressed for a 200MHz running frequency. The different legend entries refer to different parts of the design: NTT_DP to the three NTT data paths, NTT_TW to the two NTT twiddle path, PWMM to the four point-wise modular multiplications, and GTW to the overall twiddle path (GENTW + GENITW + GENPCTW). The artificial latency for synchronization between the twiddle and data path have not been taken into account.

Conclusions and perspectives

Retrospective

This thesis contributed to the preservation of privacy during data processing. In particular, it addressed the problematic of low execution performances of encrypted-computing with homomorphic cryptography by the exploration of dedicated computing architectures.

In a first chapter, we presented our position within the homomorphic cryptography research area. We identified the RLWE-based HE schemes as a promising family, well-accepted by the FHE community. In particular, we chose to focus on L-FHE schemes like FV and SHIELD as they appeared at the beginning of this thesis more mature than the fourth generation's scheme TFHE. The main issue in implementing these schemes remains the management of wide ranges of parameters that makes the definition of a hardware acceleration strategy more difficult. After having reviewed the techniques and questions for the concrete use of these schemes, a state-of-the-art of hardware implementation was presented. Our analysis of the latter concluded with our objective of defining a scalable hardware acceleration strategy with respect to the variety of parameter sets, and with our choice of specifically studying the FV scheme for that purpose.

Chapter 2 presented our analysis of the FV scheme towards its hardware acceleration. It first presented the scheme while highlighting the complexity of its parameterization. We then analyzed its requirements for the acceleration of polynomial ring multiplications by making a performance profiling of a typical encrypted application. Our analysis of the related works for the acceleration of these operations led us to choose an acceleration strategy based on coupling the RNS and the NTT approaches. After having presented both RNS and NTT utilization in this context, we validated the feasibility of their coupling for up to very large parameter sets. We then incorporated it into the dynamics of the literature by describing a full RNS variant of FV, concomitant with our work. The chapter was concluded with details on the full RNS variant profiling, indicating the relevance of accelerating both RNS specific functions and NTT-based residue polynomial multiplications.

Chapters 3 and 4 detailed our contributions to the design of polynomial ring multiplication architectures with the RNS/NTT coupled approach. These contributions make possible the consideration of large parameter sets, and thus the hardware acceleration for large encrypted-computing applications. In particular, they allow a more effective management of twiddle factors in this context on two aspects. Firstly, by an on-the-fly handling of the different twiddle sets feeding the data path of a multi-field NTT circuit. Secondly, by an on-the-fly generation of these twiddle sets from a greatly reduced number of seed elements. For these two contributions, we proposed an automatic generation of RTL circuits for a natural integration with the SPIRAL tool. The automation of these circuits facilitates the exploration of the design space for efficient NTT-based residue polynomial multiplications.

Finally, Chapter 5 presented our proposal of a hybrid computation system for the acceleration of the RNS/NTT coupled approach for FV. In particular, it showed the acceleration expectations for each parts of the ciphertext multiplication. The GPU acceleration of RNS specific functions gave promising timings up to 33 times (respectively 61 times) faster than on CPU for basis extension (respectively scale-and-round). Although, some additional developments on communications and on workload management would be beneficial for proper integration of the hybrid system. A proof-of-concept implementation of a residue polynomial multiplier was also presented in that chapter. The study of its scalabilty confirmed that our basic blocks presented in Chapters 3 and 4 allow the conception of high-throughput RPM designs for large parameter sets. The remaining limitations are located in the BRAMs utilization and in the communication bandwidth required to fully load the accelerator. Together, the GPU and FPGA acceleration made us project a significant acceleration for encrypted computing with FV. Our acceleration projections allow us to hope an order of magnitude of gain in the execution time for ciphertext multiplication with respect to state-of-the-art software implementations.

Perspectives

Research on efficient encrypted-computing continues. Throughout this thesis, we have sought to take into account the complexity of implementing homomorphic encryption. This complexity is firstly to position oneself within the different families and generations of HE schemes.

A lull can be felt in the literature since 2016, but many things remain to be done to clarify the benefits and use cases of each scheme. At least, this is the impression that emerges when one has to make a development effort towards hardware. It is always possible that one scheme may take precedence over the others in a slightly more radical way. In this case our thoughts go towards TFHE. Despite this, we hope that our scientific contributions will allow to take the use of homomorphic encryption for data privacy a step further.

Hardware acceleration for FV. In this thesis, we have shown the possibility of having a generation of twiddle factors for NTTs for a hardware cost independent of FV parameters as well as performing an automated generation of multi-field NTT circuits by associating SPIRAL and the twiddle path generator.

The refinement of our twiddle path generator for a closer interaction with SPIRAL's hardware backend could open efficient design space exploration for NTT-based RPM circuits. In a mid-term future, this generation could even be the source of an all-in-one IP for finite-field NTTs. That being said, it would still be necessary to carry out a more in-depth study of the various use-cases and their associated solutions (multi-field or single-field, choice of modulus, etc.).

Concerning the RPM operation, our work reduces the cost of coupling NTT and RNS together. Thus, it offers good perspectives for the concrete acceleration of encrypted-computing with FV. In particular, for the feasibility of an accelerator embedding all residue polynomial operations (RPMs and RPAs). Thus, the overall tensor product and relinearization could be performed locally on the dedicated hardware accelerator. The hardware cost overhead for data-flow RPA is rather small, but the local storage of numerous residue polynomials in that case could be problematic. This could motivate to explore external buffering solutions using DDR memories, directly accessed by the dedicated hardware, or even multi-FPGA approaches to separate the tensor product from the relinearization.

As a consequence, we have paved the way towards the design of a micro-server architecture for encrypted-computing along the lines of Chapter 5. Yet, our work only started to explore the communication problematics which would crop up in such a system. Our first choice of using PCIe interconnect could be suitable for a proof-of-concept hybrid system. This could be improved to increase communication bandwidth and hence furthemore improve the accelerations. For instance, by exploring the integration of the GPU and dedicated hardware for polynomial arithmetic closer to CPU memory. Staying in a multi-SoC approach, one could consider IBM's Coherent Accelerator Processor Interface (CAPI). A more integrated heterogeneous single-SoC may explore Intel's UltraPath Interconnect (UPI). Finally, considering an ASIC design, it would be interesting to take a closer look at some specific technologies such as 3D memory integration. This is a classic design problematic for application domains that have their computational performance more dependent on the quantity of data to handle than on the computations themselves. From this point of view, the acceleration of homomorphic encryption is similar to the acceleration of on-the-fly images and videos processing.

Hardware acceleration for TFHE. The implementation issues of the TFHE scheme are quite different from those of FV. Indeed, the complexity of large parameterization dynamic is less present.

However, computational complexity is also related to the ability of performing efficient polynomial multiplications. The polynomials have real coefficients modulo 1, and therefore a hardware acceleration by Fourier transform could also be considered. Consistently with our approach, the SPIRAL's DFT generator could once again be a good starting point which hints that even simplified versions of the accelerators designed in this thesis may be sufficient to accelerate TFHE in hardware.

Post-quantum cryptography. After the call from the National Institute of Standards and Technology (NIST) for post-quantum cryptography, we note that some candidates are using the algebraic structure of polynomial rings. For example, a preliminary analysis on the CRYSTAL-KYBER proposal [102] hints that this cryptosystem could benefit from NTT-based polynomial ring multiplications. This would especially be important for high-performances encryption server e.g. at the corporate end of a VPN.

In this case, the derivation of the parameters is very different from the homomorphic context. Our handling of twiddle factor sets would not be required due to sufficiently small constant modulus ($q = 7681$ for CRYSTAL-KYBER). However, as observed for elliptic-curve cryptography [103], the RNS could be useful to obfuscate manipulations of the secret against side-channel attacks, or simply to accelerate the primitives by involving multi-key parallelism. Thus our work on RNS/NTT coupling could be a starting point for the exploration of efficient hardware implementation strategy.

With or without RNS, the adaptation of the hardware backend of SPIRAL, as was done in this thesis, could generate efficient NTT circuits for a high-performance post-quantum cryptography.

Personal bibliography

Publications

J. Cathébras, A. Carbon, P. Milder, R. Sirdey, N. Ventroux. **Data Flow Oriented Hardware Design of RNS-based Polynomial Multiplication for SHE Acceleration**. *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(3), 69-88.*

J. Cathébras, A. Carbon, R. Sirdey, N. Ventroux. **An Analysis of FV Parameters Impact Towards Its Hardware Acceleration**. *5th Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC 2017).* In: Brenner M. et al. (eds) Financial Cryptography and Data Security. FC 2017. Lecture Notes in Computer Science, vol 10323, 91-106. Springer, Cham.

Patents

J. Cathébras, A. Carbon, R. Sirdey, N. Ventroux. **Processeur NTT par flot**. Numéro de dépot INPI 1856351.

J. Cathébras, A. Carbon, R. Sirdey, N. Ventroux. **Circuit de génération de facteurs de rotation pour processeur NTT**. Numéro de dépot INPI 1856340.

Communications

Conference on Cryptographic Hardware and Embedded Systems (CHES 2018), *september 2018, Amsterdam, Netherlands.* Presentation:
Data Flow Oriented Hardware Design of RNS-based Polynomial Multiplication for SHE Acceleration.

Colloque du GDR SoC/SiP, *june 2017, Bordeaux, France.* Poster:
An Analysis of FV Parameters Impact Towards Its Hardware Acceleration.

5th Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC 2017),*april 2017, Sliema, Malta.* Presentation:
An Analysis of FV Parameters Impact Towards Its Hardware Acceleration.

Journées Codage & Cryptographie, *april 2017, La Bresse, France.* Presentation:
An Analysis of FV Parameters Impact Towards Its Hardware Acceleration.

Architectures des Systèmes Matériels et Logiciels Embarqués et Méthodes de Conception Associées, *march 2017, Nancy, France*. Poster:
An Analysis of FV Parameters Impact Towards Its Hardware Acceleration.

Bibliography

[1] European Union Agency for Network and Information Security (ENISA). Data Protection - Privacy by Design. https://www.enisa.europa.eu/topics/data-protection/privacy-by-design. Accessed: 2018-10-07.

[2] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[3] J. Stern. *La science du secret*. Sciences. Editions Odile Jacob, 1998.

[4] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes. In *Selected Areas in Cryptography - SAC*, St. John's, Newfoundland and Labrador, Canada, August 2016.

[5] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2018/117, 2018. https://eprint.iacr.org/2018/117.

[6] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption over the Torus. Cryptology ePrint Archive, Report 2018/421, 2018. https://eprint.iacr.org/2018/421.

[7] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[8] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, sep 2016.

[9] Guillaume Bonnoron. *A Journey Towards Practical Homomorphic Encryption*. PhD thesis, Université de Bretagne Sud, 2018.

[10] Franz Franchetti, José Moura, and Markus Püschel. Spiral - Software/Hardware Generation for Performance. http://www.spiral.net. Accessed: 2018-10-07.

[11] International Telecommunication Union. ICT Facts and Figures 2015. https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf. Accessed: 2018-10-07.

[12] International Telecommunication Union. ICT Facts and Figures 2017. https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf. Accessed: 2018-10-07.

[13] Bastien Lepine. Le Big Data, l'IA et le Machine Learning transforment les soins de santé. https://www.lebigdata.fr/big-data-soins-de-sante. Accessed: 2018-10-07.

[14] Bastien Lepine. Enfants et Big Data : la collecte de données, un danger selon l'UNICEF. https://www.lebigdata.fr/enfants-big-data-danger. Accessed: 2018-10-07.

[15] Commission Nationale de l'Informatique et des Libertés (CNIL). Loi 78-17 du 6 janvier 1978 modifiée. https://www.cnil.fr/fr/loi-78-17-du-6-janvier-1978-modifiee. Accessed: 2018-10-07.

[16] Commission Nationale de l'Informatique et des Libertés (CNIL). Le contrôle de l'utilisation d'internet et de la messagerie électronique. https://www.cnil.fr/fr/le-controle-de-lutilisation-dinternet-et-de-la-messagerie-electronique. Accessed: 2018-10-07.

[17] 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*. ACM Press, 1982.

[18] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, apr 1984.

[19] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[20] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, pages 223–238. Springer Berlin Heidelberg.

[21] T. Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, jul 1985.

[22] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography*, pages 325–341. Springer Berlin Heidelberg, 2005.

[23] 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.

[24] 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.

[25] Craig Gentry et al. Fully Homomorphic Encryption using Ideal Lattices. In *STOC*, volume 9, pages 169–178, May 2009.

[26] Nigel P Smart and Frederik Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *Public Key Cryptography–PKC 2010*, pages 420–443. Springer, Jan 2010.

[27] Craig Gentry and Shai Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In *Advances in Cryptology–EUROCRYPT 2011*, pages 129–148. Springer, February 2011.

[28] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, oct 2011.

[29] Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[30] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, Jan 2012.

[31] Zvika Brakerski. Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.

[32] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[33] Jacob Alperin-Sheriff and Chris Peikert. Faster Bootstrapping With Polynomial Error. In *Advances in Cryptology–CRYPTO 2014*, pages 297–314. Springer, 2014.

[34] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[35] Jean-François Biasse and Luis Ruiz. FHEW with Efficient Multibit Bootstrapping. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, pages 119–135, Berlin, Heidelberg, 2015. Springer-Verlag.

[36] 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, New York, NY, USA, 2005. ACM.

[37] Oded Regev. The Learning with Errors Problem (Invited Survey). In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, pages 191–204, Washington, DC, USA, 2010. IEEE Computer Society.

[38] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[39] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.

[40] Yarkın Doröz and Berk Sunar. Flattening NTRU for Evaluation Key Free Homomorphic Encryption. Cryptology ePrint Archive, Report 2016/315, 2016. https://eprint.iacr.org/2016/315.

[41] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[42] Damien Stehlé and Ron Steinfeld. Making NTRU as Secure as Worst-Case Problems over Ideal Lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 27–47, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[43] Martin Albrecht, Shi Bai, and Léo Ducas. A Subfield Lattice Attack on Overstretched NTRU Assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 153–178, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[44] Paul Kirchner and Pierre-Alain Fouque. Revisiting Lattice Attacks on Overstretched NTRU Parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 3–26, Cham, 2017. Springer International Publishing.

[45] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Advances in cryptology–EUROCRYPT 2010*, pages 24–43. Springer, May 2010.

[46] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In *Advances in Cryptology–CRYPTO 2011*, pages 487–504. Springer, 2011.

[47] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology–EUROCRYPT 2012*, pages 446–464. Springer, April 2012.

[48] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-Invariant Fully Homomorphic Encryption over the Integers. In *Public-Key Cryptography–PKC 2014*, pages 311–328. Springer, 2014.

[49] Jung Hee Cheon and Damien Stehlé. Fully Homomophic Encryption over the Integers Revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 513–536, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[50] Chris Peikert. Public-key Cryptosystems from the Worst-case Shortest Vector Problem: Extended Abstract. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

[51] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical Hardness of Learning with Errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM.

[52] Vincent Migliore. *Hardware Cybersecurity and Design of Dedicated Components for the Acceleration of Homomorphic Encryption Schemes.* PhD thesis, Université de Bretagne Sud, September 2017.

[53] Martin Albrecht. lwe-estimator, Sage Module for Estimating the Concrete Security of LWE Instances.

[54] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors over Rings*, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[55] Chris Peikert. How Not to Instantiate Ring-LWE. In *Proceedings of the 10th International Conference on Security and Cryptography for Networks - Volume 9841*, pages 411–430, Berlin, Heidelberg, 2016. Springer-Verlag.

[56] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of Ring-LWE for Any Ring and Modulus. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 461–473, New York, NY, USA, 2017. ACM.

[57] N.P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. Cryptology ePrint Archive, Report 2011/133, 2011. https://eprint.iacr.org/2011/133.

[58] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully Homomorphic Encryption with Polylog Overhead. Cryptology ePrint Archive, Report 2011/566, 2011. https://eprint.iacr.org/2011/566.

[59] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.

[60] Pierrick Méaux. *Hybrid Fully Homomorphic Framework*. PhD thesis, PSL Research University, December 2017.

[61] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In *Fast Software Encryption*, pages 313–333. Springer Nature, 2016.

[62] Angela Jäschke and Frederik Armknecht. (Finite) Field Work: Choosing the Best Encoding of Numbers for FHE Computation. Cryptology ePrint Archive, Report 2017/582, 2017. https://eprint.iacr.org/2017/582.

[63] Kalpana Singh, Renaud Sirdey, and Sergiu Carpov. Practical Personalized Genomics in the Encrypted Domain. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, apr 2018.

[64] Kim Laine, Hao Chen, and Rachel Player. Simple Encrypted Arithmetic Library. https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library. Accessed: 2018-08-24.

[65] Yuriy Polyakov, Kurt Rohloff, and Gerard Ryan. PALISADE Lattice Cryptography Library. https://git.njit.edu/palisade/PALISADE. Accessed: 2018-08-24.

[66] Tancrède Lepoint. FV-NFLlib. https://github.com/CryptoExperts/FV-NFLlib. Accessed: 2018-08-24.

[67] Pascal Aubry, Sergiu Carpov, Paul Dubrulle, Simon Fau, Vincent Herbert, Malika Izabachène, Thanh-Hai Nguyen, Donald Nokam-Kuate, Patrick Ruf, Kalpana Singh, Oana Stan, and Renaud Sirdey. Cingulata: Homomorphic Cryptography Compiler Toolchain and Runtime Environment. https://github.com/CEA-LIST/Cingulata. Accessed: 2018-08-24.

[68] Alhassan Khedr and Glenn Gulak. SecureMed: Secure Medical Computation Using GPU-Accelerated Homomorphic Encryption Scheme. 22:597–606, mar 2018.

[69] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. 145:293–294, 1962.

[70] Stephen. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, 1966.

[71] John M Pollard. The Fast Fourier Transform in a Finite Field. *Mathematics of computation*, 25(114):365–374, 1971.

[72] Huapeng Wu. On Computation of Polynomial Modular Reduction. Technical report, The Center for Applied Cryptographic Research, University of Waterloo, 2000.

[73] Peter L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–519, may 1985.

[74] Paul Barrett. Implementing the Rivest, Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology — CRYPTO' 86*, pages 311–323. Springer Berlin Heidelberg.

[75] A. Mkhinini, P. Maistri, R. Leveugle, R. Tourki, and M. Machhout. A flexible RNS-based large polynomial multiplier for Fully Homomorphic Encryption. In *2016 11th International Design & Test Symposium (IDT)*. IEEE, dec 2016.

[76] Vincent Migliore, Maria Mendez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, and Guy Gogniat. Hardware/Software Co-Design of an Accelerator for FV Homomorphic Encryption Scheme Using Karatsuba Algorithm. *IEEE Transactions on Computers*, 67(3):335–347, mar 2018.

[77] Thomas Pöppelmann, Michael Naehrig, Andrew Putnam, and Adrian Macias. Accelerating Homomorphic Evaluation on Reconfigurable Hardware. In *Lecture Notes in Computer Science*, pages 143–163. Springer Berlin Heidelberg, 2015.

[78] Sujoy Sinha Roy, Kimmo Järvinen, Frederik Vercauteren, Vassil Dimitrov, and Ingrid Verbauwhede. Modular Hardware Architecture for Somewhat Homomorphic Function Evaluation. In *Lecture Notes in Computer Science*, pages 164–184. Springer Berlin Heidelberg, 2015.

[79] Erdinç Öztürk, Yarkin Doröz, Berk Sunar, and Erkay Savas. Accelerating Somewhat Homomorphic Evaluation using FPGAs. *IACR Cryptology ePrint Archive*, 2015:294, 2015.

[80] Erdinc Ozturk, Yarkin Doroz, Erkay Savas, and Berk Sunar. A Custom Accelerator for Homomorphic Encryption Applications. *IEEE Trans. Comput.*, 66(1):3–16, January 2017.

[81] David Bruce Cousins, Kurt Rohloff, and Daniel Sumorok. Designing an FPGA-Accelerated Homomorphic Encryption Co-Processor. *IEEE Transactions on Emerging Topics in Computing*, 5(2):193–206, apr 2017.

[82] M. Khairallah and M. Ghoneima. Tile-Based Modular Architecture for Accelerating Homomorphic Function Evaluation on FPGA. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, Oct 2016.

[83] Donald Donglong Chen, Nele Mentens, Frederik Vercauteren, Sujoy Sinha Roy, Ray C.C. Cheung, Derek Pao, and Ingrid Verbauwhede. High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems. Cryptology ePrint Archive, Report 2014/646, 2014. https://eprint.iacr.org/2014/646.

[84] Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *Progress in Cryptology–AFRICACRYPT 2014*, pages 318–335. Springer, 2014.

[85] Adeline Langlois and Damien Stehlé. Hardness of Decision (R)LWE for any Modulus. *IACR Cryptology ePrint Archive*, 2012:91, 2012.

[86] 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*. Association for Computing Machinery (ACM), 2015.

[87] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. http://gmplib.org/.

[88] W. B. Hart. Fast Library for Number Theory: An Introduction. In *Proceedings of the Third International Congress on Mathematical Software*, ICMS'10, pages 88–91, Berlin, Heidelberg, 2010. Springer-Verlag. http://flintlib.org.

[89] Richard Lindner and Chris Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. Cryptology ePrint Archive, Report 2010/613, 2010. https://eprint.iacr.org/2010/613.

[90] Martin R. Albrecht, Rachel Player, and Sam Scott. On the Concrete Hardness of Learning With Errors. *Journal of Mathematical Cryptology*, 9(3), jan 2015.

[91] Martin R. Albrecht. On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL. In *Lecture Notes in Computer Science*, pages 103–129. Springer International Publishing, 2017.

[92] Christophe De Canniere and Bart Preneel. TRIVIUM Specifications. *eSTREAM, ECRYPT Stream Cipher Project*, 2006.

[93] Nicholas Nethercote, Robert Walsh, and Jeremy Fitzhardinge. "building workload characterization tools with valgrind". In *2006 IEEE International Symposium on Workload Characterization*. Institute of Electrical and Electronics Engineers (IEEE), oct 2006.

[94] Jean-Claude Bajard and Thomas Plantard. RNS bases and conversions. In Franklin T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations XIV*. SPIE, oct 2004.

[95] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-Based Fast Lattice Library. In *Topics in Cryptology - CT-RSA 2016*, pages 341–356. Springer Nature, 2016.

[96] David Harvey. Faster Arithmetic for Number-Theoretic Transforms. *Journal of Symbolic Computation*, 60:113–119, jan 2014.

[97] Peter Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. Computer Generation of Hardware for Linear Digital Signal Processing Transforms. *ACM Transactions on Design Automation of Electronic Systems*, 17(2):1–33, apr 2012.

[98] Lingchuan Meng. *Automatic Library Generation and Performance Tuning for Modular Polynomial Multiplication.* PhD thesis, Drexel University, 2015.

[99] Marcela Zuluaga, Peter Milder, and Markus Püschel. Streaming Sorting Networks. *ACM Trans. Des. Autom. Electron. Syst.*, 21(4):55:1–55:30, May 2016.

[100] Peter A. Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. Discrete Fourier Transform Compiler: From Mathematical Representation to Efficient Hardware. CSSI Technical Report CSSI-07-01, Carnegie Mellon University, 2007.

[101] Markus Püschel, Peter A. Milder, and James C. Hoe. Permuting Streaming Data Using RAMs. *Journal of the ACM*, 56(2):1–34, apr 2009.

[102] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, apr 2018.

[103] Karim Bigou. *Etude théorique et implantation matérielle d'unités de calcul en représentation modulaire des nombres pour la cryptographie sur courbes elliptiques.* PhD thesis, Université de Rennes 1, 2014. Français.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Accélération matérielle pour la cryptographie homomorphe

Mots clés : Protection des données, Cryptographie homomorphe, Accélération matérielle

Résumé : Dans cette thèse, nous nous proposons de contribuer à la définition de systèmes de crypto-calculs pour la manipulation en aveugle de données confidentielles. L'objectif particulier de ce travail est l'amélioration des performances du chiffrement homomorphe. La problématique principale réside dans la définition d'une approche d'accélération qui reste adaptable aux différents cas applicatifs de ces chiffrements, et qui, de ce fait, est cohérente avec la grande variété des paramétrages. C'est dans cet objectif que cette thèse présente l'exploration d'une architecture hybride de calcul pour l'accélération du chiffrement de Fan et Vercauteren (FV).
Cette proposition résulte d'une analyse de la complexité mémoire et calculatoire du crypto-calcul avec FV. Une partie des contributions rend plus efficace l'adéquation d'un système non-positionnel de représentation des nombres (RNS) avec la multiplication de polynôme par transformée de Fourier sur corps finis (NTT). Les opérations propres au RNS, facilement parallélisables, sont accélérées par une unité de calcul SIMD type GPU. Les opérations de NTT à la base des multiplications de polynôme sont implémentées sur matériel dédié de type FPGA. Des contributions spécifiques viennent en soutien de cette proposition en réduisant le coût mémoire et le coût des communications pour la gestion des facteurs de rotation des NTT.
Cette thèse ouvre des perspectives pour la définition de micro-serveurs pour la manipulation de données confidentielles à base de chiffrement homomorphe.

Title : Hardware acceleration for homomorphic encryption

Keywords : Data Privacy, Homomorphic Cryptographie, Hardware Acceleration

Abstract : In this thesis, we propose to contribute to the definition of encrypted-computing systems for the secure handling of private data. The particular objective of this work is to improve the performance of homomorphic encryption. The main problem lies in the definition of an acceleration approach that remains adaptable to the different application cases of these encryptions, and which is therefore consistent with the wide variety of parameters. It is for that objective that this thesis presents the exploration of a hybrid computing architecture for accelerating Fan and Vercauteren's encryption scheme (FV).
This proposal is the result of an analysis of the memory and computational complexity of crypto-calculation with FV. Some of the contributions make the adequacy of a non-positional number representation system (RNS) with polynomial multiplication Fourier transform over finite-fields (NTT) more effective. RNS-specific operations, inherently embedding parallelism, are accelerated on a SIMD computing unit such as GPU. NTT-based polynomial multiplications are implemented on dedicated hardware such as FPGA. Specific contributions support this proposal by reducing the storage and the communication costs for handling the NTTs' twiddle factors.
This thesis opens up perspectives for the definition of micro-servers for the manipulation of private data based on homomorphic encryption.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France