Lattice-based digital signature and discrete gaussian sampling

Thomas Ricosset

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Réseaux, Télécommunications, Systèmes et Architecture

Présentée et soutenue par :

M. THOMAS RICOSSET

le lundi 12 novembre 2018

Titre :

Signature électronique basée sur les réseaux euclidiens et échantillonnage selon une loi normale discrète

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. CARLOS AGUILAR MELCHOR

Rapporteurs :

M. DAMIEN STEHLE, ECOLE NORMALE SUP LYON ENS DE LYON
M. DUONG-HIEU PHAN, UNIVERSITE DE LIMOGES

Membre(s) du jury :

M. PHILIPPE GABORIT, UNIVERSITE DE LIMOGES, Président
M. CARLOS AGUILAR MELCHOR, INP TOULOUSE, Membre
M. EMMANUEL CHAPUT, INP TOULOUSE, Membre
Mme ADELINE LANGLOIS, CNRS, Membre

Lattice-Based Digital Signature and Discrete Gaussian Sampling

Thomas RICOSSET

Supervisor: Carlos AGUILAR MELCHOR

Remerciements

Mes premiers remerciements vont à mon directeur de thèse Carlos Aguilar Melchor. En premier lieu, pour son encadrement de qualité qui m'a enseigné, d'abord la rigueur, puis l'autonomie, dans le métier de chercheur. Ensuite, pour m'avoir proposé ce sujet de thèse, aussi passionnant que porteur et pour son intarissable enthousiasme qui a incontestablement eu un impact positif sur ma recherche. Enfin, pour toutes nos discussions intéressantes qui ont forgé ma vision de la recherche académique.

J'exprime maintenant ma plus grande gratitude envers Alexandre Anzala Yamajako, Sylvain Lachartre et Eric Garrido, qui m'ont donné l'opportunité d'effectuer ma thèse en milieu industriel dans les meilleures conditions, ont partagé avec moi leur expertise pointue dans differents domaines de la cryptologie et m'ont soutenu dans toutes sortes de projets, qu'ils soient professionnels ou personnels. Leur bienveillance est une motivation constante à donner le meilleur de soi-même, ainsi qu'un plaisir à vivre. Merci beaucoup Alexandre et Sylvain de m'avoir donné ma chance en stage, puis de m'avoir recommandé pour cette thèse !

Je suis infiniement reconnaissant à Duong-Hieu Phan et Damien Stehlé qui ont rédigé des rapports réfléchis et pertinants de mon manuscrit de thèse malgré des emplois du temps très chargés. J'admire profondément leurs travaux de recherche et je me sens extrémement chanceux qu'ils aient accepté de lire, commenter et approuver mon manuscrit. J'aimerais également remercier Emmanuel Chaput, Philippe Gaborit et Adeline Roux-Langlois qui ont accepté de faire partie de mon jury de thèse et avec qui j'ai partagé de plaisantes et intéréssantes discussions.

Je tiens également à remercier tous mes co-auteurs : Carlos Aguilar Melchor, Martin R. Albrecht, Jean-Christophe Deneuville, Pierre-Alain Fouque, Philippe Gaborit, Jeffrey Hoffstein, Marc-Olivier Killijian, Paul Kirchner, Tancrède Lepoint, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Gregor Seiler, William Whyte et Zhenfei Zhang, auprès de qui j'ai beaucoup appris et avec qui j'ai apprécié travailler. Je remercie en particulier Thomas Prest de m'avoir convié à cette grande avanture qu'est Falcon et l'appel à candidature du NIST et qui a partagé avec moi, tout au long de ces dernières années, son expertise pointue de la cryptographie post-quantique et en particulier à base de réseaux euclidiens.

J'ai été particulièrement privilégié de faire ma thèse au sein du laboratoire Chiffre de Thales, qui offre un environnement de travail idéal, une très grande liberté et un soutient exceptionnel quant aux travaux de recherche réalisés, en présence de personnes dont les compétences et la gentillesse continuent de me frapper chaque jour davantage. Merci aux présents : Ange, Anne, Aurélien, David, Didier, Emeline, Eric, Julien, Melissa, Mickael, Olivier, Olivier, Philippe, Renaud, Simon, Sylvain, Thomas et à ceux qui sont partis et que j'ai connus : Alexandre, Julia, Lydie et Sonia, pour ces excellentes années et merci de m'offrir une place parmi vous pour la suite.

Je remercie également toute l'équipe RMESS de l'Institut de Recherche en Informatique de Toulouse pour l'ambiance toujours agréable au laboratoire et l'entraide entre les doctorants. Merci donc à Adrien, André-Luc, Béatrice, Cédric, Christian, Emmanuel, Gentian, Jean-Luc, Julien, Riadh, Katia,

Oana, Samer, Sylvie, Yoann et les autres. J'ai eu trop peu de temps à passer avec vous, mais j'en ai apprécié chaque moment. Merci en particulier à Cédric, Oana et tous ceux qui ont partagé avec moi le bureau F405.

Je suis très reconnaissant aussi à toutes ces rencontres aux cours de differentes conférences et séminaires qui rendent la communauté cryptographique si attachante. Je pense notamment au soutien initial de Tancrède, aux discussions captivantes avec Adeline, Benoit, David, Fabrice, Julien, Léo, Martin, Mehdi et je n'oublie bien sûr pas tous ces moments chaleureux passés avec eux mais aussi avec d'autres doctorants : Fabrice, Geoffroy, Houda, Pierrick, Raphael et Remi.

Merci aussi à tous ceux qui ont participé de manière indirecte à cette avanture : À ma mère, pour mon inaliénable éducation, ainsi que pour son indéfectible soutien tout au long de ces nombreuses années. À mon père, qui a su cultiver en moi le goût du savoir et une passion certaine pour les systèmes d'information. À mes grand-parents, oncles et tantes, qui ont toujours été pour moi, tous à leur façon, de précieux modèles d'accomplissement personnel. À Clémence qui me rend heureux. Et à mes amis les plus proches Alexandre, Charlotte, Dine, Laura, Léonard, Mickaël, Samy, Valentin et Stéphane qui me rendent la vie très gaie.

Un savant dans son laboratoire n'est pas seulement un technicien : c'est aussi un enfant placé devant des phénomènes naturels qui l'impressionnent comme des contes de fées. Nous devons avoir un moyen pour communiquer ce sentiment à l'extérieur, nous ne devons pas laisser croire que tout progrès scientifique se réduit à des machines et des engrenages.

Marie Curie

Résumé en Français

Cette thèse concerne un domaine de la cryptographie au carrefour de la théorie des nombres, de la géométrie algébrique et de la théorie de la complexité. La cryptographie est une discipline s'attachant à protéger des données (ou messages) assurant en particulier leur confidentialité, intégrité et authenticité. Ainsi, la cryptographie comprend des techniques dites de chiffrement, rendant un message inintelligible à autre que qui-de-droit, et dites de signature, permettant de vérifier que le message n'ait pas été altéré et qu'il provienne de l'émetteur présumé.

La sécurité de la cryptographie usuelle repose sur des problèmes mathématiques réputés difficiles à résoudre : trouver un logarithme discret (DLOG), ou décomposer de grands nombres en facteurs premiers (FACT). Toutefois, cette cryptographie est menacée par une percée technologique : l'ordinateur quantique. En effet, les opérations élémentaires des ordinateurs quantique et classiques étant de natures différentes elles rendent ce premier capable de résoudre les problèmes DLOG et FACT efficacement. Cette menace a fait s'intéresser les chercheurs à des alternatives pouvant lui résister.

L'une de ces alternatives, parmi les plus prometteuses de cette décennie, repose sur des problèmes géométriques d'un objet mathématique connu sous le nom de réseau euclidien. Outre sa potentielle résistance à l'ordinateur quantique, la cryptographie à base de réseaux euclidiens ouvre également de nouvelles perspectives en permettant d'effectuer de multiples opérations sur les données chiffrés sans avoir à les déchiffrer. Cependant, bon nombre de constructions sur les réseaux euclidiens nécessitent un générateur de bruit gaussien à haute précision. Ces générateurs sont essentiels à la sécurité des schémas qui les abritent, mais limite souvent leur efficacité.

Ce résumé comprend une introduction à cette cryptographie, reprenant en partie l'article de vulgarisation intitulé *Une Cryptographie Nouvelle : Le Réseau Euclidien* écrit par Léo Ducas-Binda et publié dans *GNU/Linux Magasin n°178*, ainsi qu'un résumé des contributions de cette thèse.

Sommaire

Des réseaux euclidiens à la cryptographie

Les réseaux euclidiens ont d'abord été utilisés pour casser certains schémas cryptographiques avant de se rendre compte qu'ils pouvaient aussi servir à en construire de nouveaux. Miklós Ajtai fut pionnier de cette nouvelle discipline en démontrant dans ces travaux de 1996 que les réseaux euclidiens peuvent servir de base solide à la cryptographie. Plus précisément, il démontra qu'il n'existe pas d'instance faible pour les problèmes sur les réseaux, c'est-à-dire que ces problèmes sont, à paramètres fixés, aussi difficiles pour tout réseau considéré, contrairement au problème FACT qui, par exemple, admet certains choix de facteurs premiers peu judicieux rendant la factorisation facile. De plus, le fait que les problèmes de réseaux euclidiens interviennent dans de nombreux domaines en informatique depuis près d'un demi-siècle sans que cela n'ait permis de les résoudre efficacement semble une garantie supplémentaire quant à la sécurité des schémas cryptographiques basés sur ces problèmes.

Portés par les idées d'Ajtai, des schémas cryptographiques basés sur les réseaux euclidiens ont rapidement été développés comme le schéma de chiffrement NTRU et celui de signature NTRUsign, mais ce dernier subie plusieurs attaques malgré les différentes tentative pour le rendre plus résistant. Il faudra attendre les années 2000 et les efforts considérables de plusieurs chercheurs pour surmonter les nombreuses difficultés théoriques liées à l'utilisation de ce nouvel objet en cryptographie et particulièrement en signature. Mais si cette étude théorique a permis de révéler le potentiel de cette cryptographie nouvelle, elle a également laissé de coté les questions d'efficacité produisant des schémas cryptographiques inutilisables en pratique bien que théoriquement sûrs. En parallèle, cette cryptographie des réseaux étonne la communauté en résolvant pour la première fois un problème considéré comme le Graal de la cryptographie en permettant d'obtenir un schéma de chiffrement dit pleinement homomorphe, c'est-à-dire permettant d'exécuter n'importe quel programme sur les données chiffrées sans devoir au préalable les déchiffrer, ouvrant ainsi la voie à un monde de possibilités nouvelles.

Depuis quelques années, la question de l'efficacité de cette cryptographie est redevenue pertinente et le NIST, organisme américain chargé de la standardisation de schémas cryptographiques, a lancé la course au post-quantique avec un appel international pour la création de standards à l'épreuve de l'ordinateur quantique. Dans cette compétition les schémas cryptographiques basés sur les réseaux euclidiens représentent plus de 40% des candidats, il ne serait alors pas étonnant de voir cette cryptographie nouvelle profiter de sa grande flexibilité afin de prendre, dans quelques années, la place de cryptographie usuelle. Cependant, l'écart existant entre instances prouvées sûres et instances utilisables en pratique représente encore aujourd'hui un défi de taille pour cette cryptographie. En effet, la taille de certains paramètres nécessaire pour garantir une sécurité prouvée est trop élevée, ce qui conduit à leurs préférer des paramètres empiriques basés sur les attaques les plus efficaces en pratique. Notons que ce processus de détermination empirique de paramètres correspond à ce qui se fait aujourd'hui pour la mise-en-œuvre de la cryptographie, mais que cette méthode nécessite encore, pour être fiable dans le contexte de la cryptographie à base de réseaux euclidien, un effort considérable de recherche d'attaques optimales.

La géométrie des réseaux euclidiens

Un réseaux euclidien est défini comme un sous-groupe discret d'un espace vectoriel euclidien. Dit autrement, un réseaux euclidien est simplement un ensemble de points arrangés régulièrement dans l'espace. La figure 1 représente un réseau euclidien de dimension deux. Plus précisément, quand nous disons arrangés régulièrement nous parlons de la structure additive de cet ensemble de points.

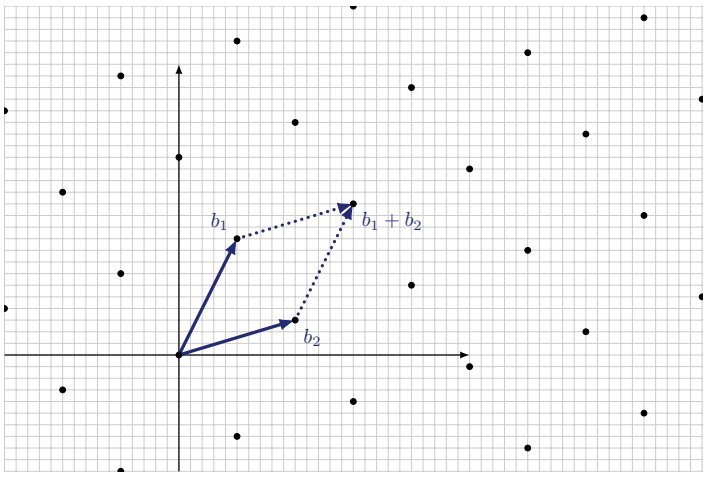


Figure 1. – Réseau euclidien (points noirs) en dimension 2 et exemple d'addition de points.

En effet, un réseaux euclidien est un ensemble de points d'un espace que l'on peut additionner ou soustraire entre eux[1]. La notion de discret s'oppose elle à celle de continu, une droite passant par l'origine aurait aussi cette structure additive, mais la droite est un ensemble continu en ce sens qu'au voisinage d'un point de la droite nous trouverons toujours un autre point de droite et ce aussi petit que soit le voisinage considéré, une telle droite n'est donc pas un réseau euclidien. Enfin, un espace vectoriel euclidien est simplement un espace droit, comme l'espace en trois dimension auquel nous sommes habitués, par opposition aux espaces courbes qu'on peut par exemple trouver dans la théorie de la relativité d'Albert Einstein.

Le nombre de points d'un réseau euclidien étant infini, il est impossible de le décrire en donnant la liste complète de ceux-ci. On le décrira donc en utilisant une base, c'est-à-dire un ensemble de points, ou vecteurs, qui engendrent entièrement le réseau euclidien, comme décrit en figure 2.

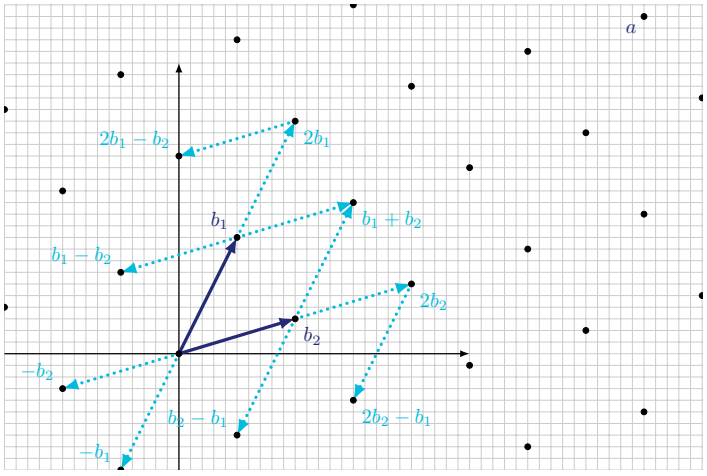


Figure 2. – Une base générant un réseau euclidien.

N'importe quel point du réseau euclidien peut être décrit par un système de coordonnées dans

[1] Additionner, respectivement soustraire, deux points de cet ensemble reviens à additionner, resp. soustraire, les vecteurs allant de l'origine du repère à ces points.

cette base. Ainsi, dans le réseau en deux dimensions de la figure 2 le point a a pour coordonnées $(2, 3)$ dans la base (b_1, b_2), ce qui signifie que $a = 2b_1 + 3b_2$. Notez que dans un réseaux euclidien les coordonnées sont toujours des nombres entiers, mais les coordonnées de la base du réseau euclidien dans base naturelle ne sont pas nécessairement des nombres entiers et celle-ci n'est pas nécessairement rectangle. En fait, en dehors de certains réseaux euclidiens très simples, il n'existera aucune base rectangle. Mais chaque réseau euclidien admet plein de bases différentes et certaines sont plus rectangles que d'autres.

Sur la figure 3, intuitivement la base bleu semble meilleure que la base rouge, elle n'est pas parfaitement rectangle, mais elle est plus rectangle que la base rouge. Nous allons voir en quoi avoir une bonne base pour un réseau est utile, mais notez que tirer un point aléatoire du réseau euclidien est facile même avec une mauvaise base simplement en choisissant aléatoirement ses coordonnées.

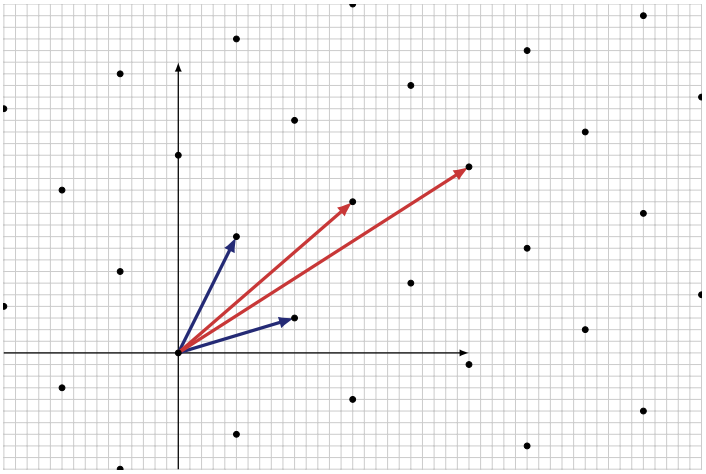


Figure 3. – Deux bases d'un même réseau euclidien, la bonne base en bleu et la mauvaise en rouge.

Pour comprendre en quoi une base est meilleure qu'une autre il faut expliquer à quoi peuvent servir les réseaux euclidiens. Lorsqu'on veut par exemple utiliser un canal analogique pour des communications numériques, on souhaite transformer le signale continu transmis en une donnée discrète, pour ce faire on peut faire correspondre la donnée analogique du signale avec un point du réseau euclidien. Cependant, il y a forcément du bruit qui vient perturber le signal, nous allons donc considérer que la donnée analogique transmise correspond à un point de l'espace complet dans lequel se plonge notre réseau euclidien. En supposant que l'erreur soit suffisamment petite, on peut retrouver la donnée numérique transmise : c'est le point du réseau euclidien le plus proche de la donnée analogique reçu. Sur la figure 4 l'émetteur cherche à transmettre le point d, mais les erreurs analogiques le transforment en $t = d + e$ pour une petite erreur e. Le récepteur retrouvera le point d qui est le point du réseau le plus proche de t.

Les bases servent à découper l'espace pour retrouver le point le plus proche comme exposé sur la figure 5. Retrouver ce point le plus proche semble facile en dimension deux en dessinant quelques cercles, mais cela se complique si on considère un réseau euclidien avec plus de dimensions. Il existe divers algorithmes dus à Laszlo Babai pour retrouver rapidement un point proche et ces algorithmes utilisent une base du réseau euclidien. Ces algorithmes pavent l'espace en fonction de la base, et chaque point du réseau euclidien est au centre de l'un de ces pavés. Comme on le voit sur la figure 5, la tolérance aux erreurs, correspondant au rayon des cercles inscrit dans chaque pavé, dépend de la forme du pavé et donc de la base. Ainsi une bonne base sera une base pour laquelle la tolérance aux

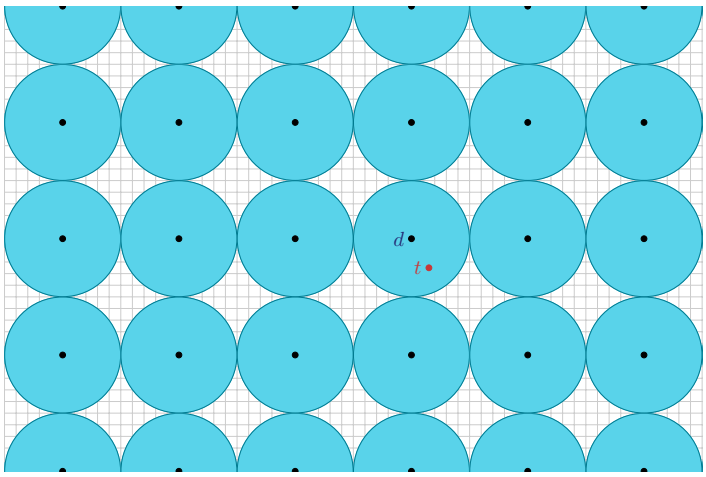


Figure 4. – Correction d'erreur utilisant un réseau carré.

erreurs est élevée.

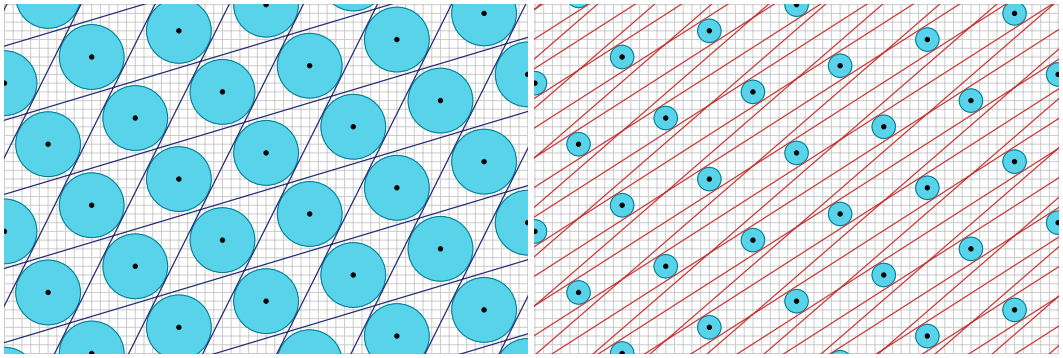


Figure 5. – Correction d'erreur avec une bonne base (à gauche) et une mauvaise base (à droite).

Lorsque le nombre de dimensions est petit trouver une bonne base pour un réseau euclidien est aisé, on parle de réduction de réseau. Mais les algorithmes de réduction de réseau s'avèrent exponentiels, ainsi, s'il est facile de réduire un réseau de dimension deux ou trois à la main, un algorithme un peu malin réduira un réseau de dimension 30 à 40 en quelques secondes et les records sur clusters atteignent péniblement 130 dimensions. Ainsi, on estime qu'avec 200 ou 250 dimensions, la réduction de réseau prendrait des milliards d'années de calculs.

Chiffrer avec les réseaux euclidiens

Les réseaux euclidiens présentent des problèmes qui sont résolubles seulement si l'on connaît une bonne base, mais qui sont difficiles sinon. Ainsi tous les ingrédients sont réunis pour construire des schémas cryptographiques basée sur une trappe. En cryptographie une trappe est une donnée qui permet de résoudre efficacement un problème qui, sans cette trappe, est difficile à résoudre. On peut se servir d'une telle trappe comme d'une clé secrète, cette clé secrète étant dans le cas des réseaux euclidiens une bonne base. Notez que la cryptographie à base de réseaux euclidiens ne fait intervenir que des calculs simple et n'implique pas de grands nombres premiers ou de structure complexe.

Pour chiffrer un message en utilisant un réseau euclidien on va considérer ce message comme une

erreur et ajouter cette erreur à un point aléatoire du réseau euclidien. Ainsi, il sera aisé de retrouver le message pour quelqu'un connaissant une bonne base, mais il sera difficile de le retrouver avec une mauvaise base et, comme vu précédemment, si la dimension du réseau euclidien est suffisamment grande il est également difficile de retrouver une bonne base à partir d'une mauvaise base. En fait, considérer le message comme une erreur n'est pas tout à fait suffisant car les messages sont généralement prédictibles. Il faut leur ajouter une erreur aléatoire supplémentaire.

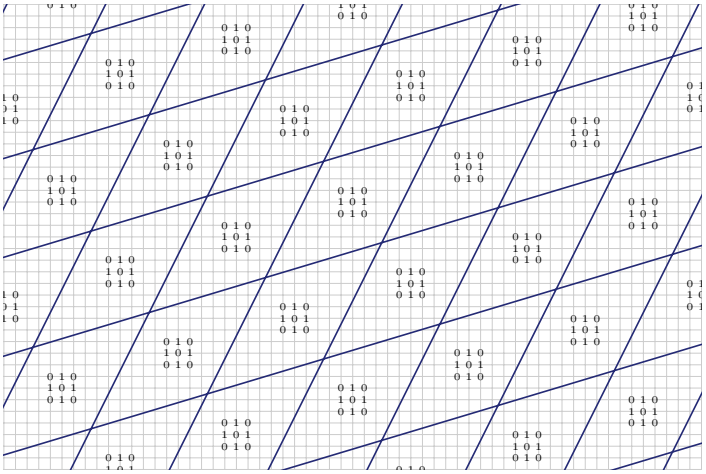


Figure 6. – Chiffrement des messages (0 ou 1) devenant des points proches des points du réseau.

Considérons l'exemple simplifié de la figure 6 où le message (0 ou 1) est chiffré en l'un des points correspondant proche d'un point du réseau euclidien. Le déchiffrement consiste alors simplement à trouver le point du réseau euclidien le plus proche, en utilisant un des algorithmes de Babai avec la bonne base, puis à le soustraire au chiffré afin de retrouver l'erreur et ainsi pouvoir conclure.

L'un de ces algorithmes de Babai est plutôt simple. Il prend en entrée la base sous forme d'une matrice et le point de l'espace, avec l'erreur, sous forme de vecteur. Ce vecteur est donnée dans le système de coordonnées naturel et on commence par le convertir vers le système de coordonnées du réseau euclidien en le multipliant par l'inverse de la matrice base. On arrondit ensuite toutes les coordonnées du vecteur obtenu et on le reconvertit vers la base naturelle en la multipliant par la matrice base. Le résultat est un point du réseau euclidien représenté par un vecteur constitué de nombres entiers.

Reste à savoir comment trouver une bonne et une mauvaise base pour un même réseau euclidien, la bonne base servant de clé privée et la mauvaise base servant de clé publique. Pour la clé privée, la bonne base, il suffit de la choisir. En effet, définir un réseau euclidien puis chercher une bonne base serait difficile, mais on peut simplement choisir une bonne base et cette base définit un réseau euclidien. Pour dériver une clé publique, une mauvaise base, à partir de cette bonne base on peut essayer de prendre une combinaison aléatoire des vecteurs de la bonne base, mais la vraie bonne solution est plus subtile, trop subtile pour être détaillée ici.

Calculer sans déchiffrer

Cette méthode de chiffrement a une propriété étonnante : si on additionne deux messages chiffrés et que l'on déchiffre le résultat, alors le résultat sera la somme des deux messages, à condition bien sûr que son erreur soit suffisamment petite (i.e. que l'erreur ne fasse pas sortir le point de son pavé engendré par la bonne base).

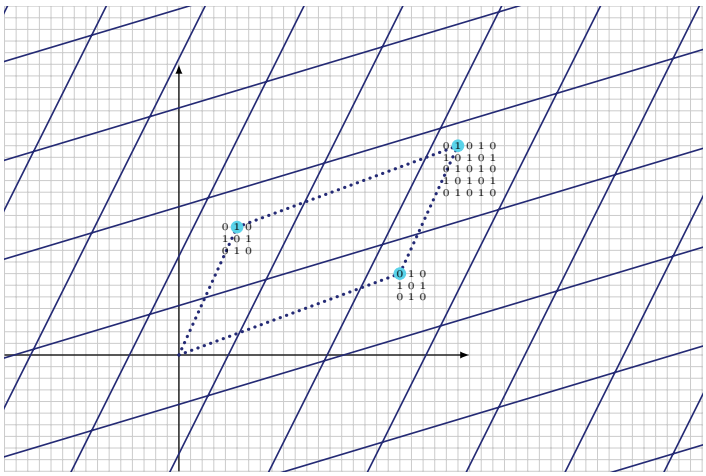


Figure 7. – Si l'erreur est petite, la somme des chiffrés est un chiffré de la somme (XOR) des clairs.

Dans la figure 7, il semble qu'effectuer une addition soit la limite : si on essaye de répéter cette opération, alors l'erreur ferait sortir le chiffré du parallélépipède bleu et le déchiffrement de celui-ci serait incorrect. Cependant cette figure est un peu trompeuse, en pratique on peut concevoir des réseaux euclidiens qui pourraient supporter plein d'addition d'affilés, mais il y a toujours une limite.

Cette propriété est appelée homomorphisme partiel, ce qui signifie que la structure est préservée, ainsi le chiffrement décrit ci-dessus préserve la structure additive du réseau euclidien et l'addition entre chiffrés se traduira par une addition des éléments chiffrés après déchiffrement sans avoir à les déchiffrer. Notez que cet homomorphisme partiel n'est pas exceptionnel, d'autres schémas cryptographiques, comme le très répandu RSA basé sur le problème FACT, ont aussi des homomorphismes partiels.

Cet homomorphisme est dit partiel car seule la structure additive est préservée. Il faudra attendre les travaux de Craig Gentry qui réussira à faire en sorte que l'on puisse aussi faire des multiplication par produit tensoriel sans déchiffrement en plus des additions, idée qui avait précédemment été étudiée par Carlos Aguilar Melchor, Philippe Gaborit et Javier Herranz. Certains peuvent se demander en quoi ceci est si remarquable, mais nous allons voir que nous disposons maintenant d'un cryptosystème qui permet d'exécuter n'importe quel programme sans avoir à déchiffrer les données.

En effet, si nous chiffrons le message bit-à-bit, c'est-à-dire si chaque bit est chiffré séparément, alors l'addition binaire (i.e. modulo 2) correspond à la porte logique XOR (i.e. ou exclusif) et la multiplication binaire correspond à la porte logique AND (i.e. et). Et vous savez sûrement que cet ensemble de deux portes logiques est universel : n'importe quel programme peut être exprimé sous forme de circuit qui n'utilise que des portes XOR et AND. On peut donc appliquer ces circuits aux données chiffrées et en théorie faire travailler un ordinateur sur des données qu'il ne peut pas comprendre.

Signer avec les réseaux euclidiens

La signature d'un message en utilisant un réseau euclidien fonctionne sur le même principe que le chiffrement. Le signataire doit disposer d'une bonne base ainsi que d'une mauvaise base d'un même réseau euclidien. Il rend alors publique la mauvaise base, celle-ci permettant à qui le souhaite de vérifier la validité des signatures et il garde privée la bonne base, lui permettant de signer des messages. Afin de signer un message, celui-ci va être transformé de manière publique et déterministe

(i.e. cette même transformation peut être effectuée par tout un chacun afin d'obtenir le même résultat) en un point de l'espace quelconque appelé point cible. Le signataire va alors, à la manière du déchiffrement vu précédemment, utiliser un des algorithmes de Babai avec sa bonne base pour trouver un point du réseau proche de ce point cible. Ce point du réseau, transmis dans le système de coordonnées naturel, fait office de signature.

Cette signature permet à chacun possédant le message et la mauvaise base de vérifier que cette signature correspond bien à la fois au signataire et au message. Cette vérification consiste en deux étapes, il faut d'abord vérifier, grâce à la mauvaise base, que la signature est bien un point du réseau euclidien, ensuite il suffit de calculer la distance qui l'éloigne du point cible. Le point cible étant propre au message et le vérifieur étant le seul à pouvoir trouver un point du réseau proche de ce point cible cela prouve bien qu'il est le signataire de ce message.

Cependant, cette méthode de signature comporte une faille dévastatrice permettant à un attaquant, disposant d'un petit nombre de signatures, de retrouver la bonne base du réseau et ainsi de forger de fausses signatures. Cela est dû au fait que si l'on soustrait le point signature au point cible le point obtenu est toujours un point du parallélépipède engendré par la bonne base (centré en l'origine), ainsi les différentes signatures vont permettre de dessiner ce parallélépipède et il sera alors aisé de retrouver la bonne base. Afin de corriger ce problème, une solution proposée par Craig Gentry, Chris Peikert et Vinod Vaikuntanathan consiste à masquer la structure de la bonne base en choisissant aléatoirement la signature parmi les points du réseau proches du point cible. Plus précisément, la distribution des signatures va alors suivre une distribution gaussienne discrète centrée en le point cible, qui est indépendante de la bonne base utilisée, ce qui garantit qu'aucune information ne fuite concernant la bonne base.

L'utilisation d'une distribution gaussienne se justifie par sa forte entropie à faible écart-type, permettant ainsi de masquer correctement la structure de la bonne base tout en conservant une taille de signature raisonnable. En effet, le vérifieur disposant du message et pouvant ainsi retrouver le point cible, seule la différente entre le point cible et le point proche du réseau est transmise comme signature. Si théoriquement l'utilisation de distributions gaussiennes discrètes corrige élégamment ce problème, il en demeure cependant un en pratique, car échantillonner selon une loi gaussienne discrète avec la précision et les différentes contraintes propres à la cryptographie n'est pas chose aisée. Et si les techniques basées sur l'échantillonnage gaussien se sont multipliées en cryptographie à base de réseaux euclidiens ces dernières années, les solutions à ce problème d'implémentation restent rares et insatisfaisantes.

Contributions de cette thèse

Dans cette section, nous présentons un résumé informel des différents travaux effectués durant mon doctorat. Chacun d'entre-eux sera développé dans la suite de la thèse.

Falcon : un nouveau schéma de signature

Falcon est un schéma de signature basé sur les réseaux euclidiens, candidat à la standardisation du NIST sur la cryptographie post-quantique, fruit d'une collaboration avec Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Gregor Seiler, William Whyte et Zhenfei Zhang. Falcon est une instanciation sur les réseaux NTRU de la méthode de signature hache-puis-signe de Gentry, Peikert et Vaikuntanathan (GPV) [GPV08] décrite précédemment. Les réseaux NTRU sont une classe particulière de réseaux euclidiens profitant d'une structure d'anneau qui rend certaines opérations beaucoup plus efficaces. Dans Falcon, nous

utilisons également une nouvelle technique d'échantillonnage gaussien sur les réseaux euclidiens que nous appelons fast Fourier sampling. En résumé, Falcon peut être décrit de manière succincte comme suit :

$$\text{Falcon} = \text{GPV} + \text{Réseaux NTRU} + \text{Fast Fourier sampling}$$

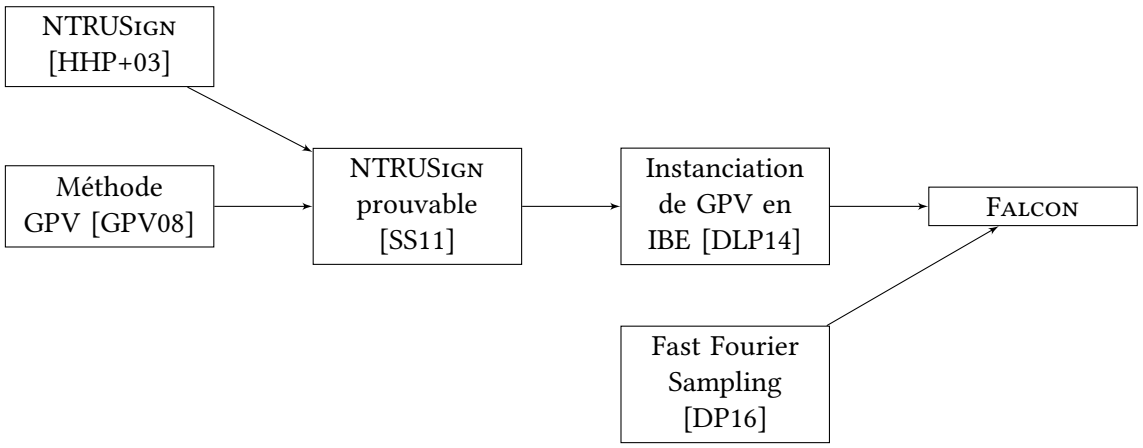


Figure 8. – L'arbre généalogique de Falcon

Généalogie de Falcon. L'arbre généalogique de Falcon commence avec le schéma de signature NTRUSign [HHP+03] de Hoffstein et al. qui fut le premier, avec GGH [GGH97], à proposer des signatures basées sur les réseaux euclidiens. L'utilisation de réseaux NTRU dans NTRUSign lui permet d'être très compacte. Cependant, comme vu précédemment ces deux schémas sont aujourd'hui cassés à cause d'une fuite d'information dans leur processus de signature qui conduit à une attaque dévastatrice permettant de retrouver la clé secrète après un petit nombre de signature [NR06; DN12b].

Comme décrit précédemment, Gentry, Peikert et Vaikuntanathan [GPV08] ont proposé une solution à cette attaque utilisant un algorithme de décodage randomisé pour calculer la signature. En plus de corriger le problème de fuite d'information dans la processus de signature, cette solution permet de disposer de schémas de signature disposant d'une sécurité prouvable. Il en résulte une méthode générale (la méthode GPV) pour construire des schémas sûrs de signature sur les réseaux euclidiens.

L'étape suivante dans l'histoire de Falcon consiste en deux travaux sur l'instanciation de la méthode GPV sur les réseaux NTRU. Le premier, dû à Stehlé et Steinfeld [SS11], consiste à combiné la méthode GPV avec les réseaux NTRU afin d'obtenir un schéma NTRUSign disposant d'une sécurité prouvable. Le second, plus pratique, dû à Ducas, Lyubashevsky et Prest [DLP14], propose une instanciation sur les réseaux NTRU et une implémentation du schéma de chiffrement basé sur l'identité (IBE) présenté avec la méthode GPV. Notons que cet IBE peut être directement converti en un schéma de signature.

Un nouvel algorithme de décodage randomisé. Après celui de la famille de réseaux, le second choix lors de l'instanciation de la méthode GPV est celui de l'algorithme de décodage randomisé. Il en existe plusieurs, chacun ayant ses avantages et ses limitations. Évidemment, l'efficacité est importante, mais il y a une métrique d'importance égale que nous nommerons qualité : le plus proche du point cible est le point renvoyé, le plus sécurisé sera le schéma de signature.

1. Algorithme de Klein [Kle00] : Variante randomisé de l'algorithme nearest plane de Babai. De bonne qualité, mais une complexité en temps quadratique en la dimension du réseau euclidien, quel que soit la structure de celui-ci.

2. Algorithme de Peikert [Pei10] : Variante randomisé de l'algorithme round-off de Babai. De complexité en temps quasi-linéaire en la dimension du réseau euclidien (au lieu de quadratique) lorsque le réseau a une structure d'anneau, mais de qualité moyenne.

3. Algorithme de Micciancio et Peikert [MP12] : Une approche nouvelle, simple et efficace, mais incompatible avec les réseaux NTRU.

4. Algorithme de Ducas et Prest [DP16] : Variante de l'algorithme nearest plane de Babai spécialement conçu pour les réseaux à structure d'anneau. Il fonctionne récursivement à la manière de l'algorithme de transformé de Fourier rapide. Cet algorithme peut être randomisé : il en résulte un générateur de bonne qualité et de complexité en temps quasi-linéaire en la dimension du réseau euclidien lorsque le réseau a une structure d'anneau.

Algorithme	Rapide	Proche de la cible	Compatible NTRU
Klein [Kle00]	**Non**	**Oui**	**Oui**
Peikert [Pei10]	**Oui**	**Non**	**Oui**
Micciancio-Peikert [MP12]	**Oui**	**Oui**	**Non**
Ducas-Prest [DP16]	**Oui**	**Oui**	**Oui**

Table 1. – Comparaison des différents algorithmes de décodage randomisé

À l'aune des quatres approches décrite ci-dessus, il nous a semblé clair qu'une varainte randomisé de l'algorithme de Ducas et Prest était le choix le mieux adapté pour Falcon étant donné le choix des réseaux NTRU. Cependant, l'instanciation pratique de cet algorithme restait une question ouverte.

Performances. Falcon s'appuie sur ces travaux pour proposer un schéma hache-puis-signe sur les réseaux euclidiens efficace en temps, mémoire et communications et profitant de fortes garanties de sécurité.

L'implémentation de référence de Falcon atteint les performances suivantes sur un processeur Intel® Core® i7-6567U (Cadencé à 3, 3 GHz), où la taille de signature correspond à la taille moyenne d'une signature compressée :

dimension	signature/s	verification/s	taille de signature
512	6081	37175	617o
768	3547	20637	994o
1024	3072	17697	1233o

Échantillonneur gaussien en centre arbitraire pour la cryptographie

Cet article intitulé *Sampling from Arbitrary Centered Discrete Gaussians for Lattice-based Cryptography* est le fruit d'une collaboration avec Carlos Aguilar-Melchor et Martin R. Albrecht et a été présenté à la conférence international *Applied Cryptography and Network Security* (ACNS) 2017.

Les schémas de signature hache-puis-signe sur les réseaux euclidiens, comme Falcon, consistent à hacher le message vers un point de l'espace avant d'utiliser un algorithme de décodage pour retrouver un point proche du réseau. Ces algorithmes de décodage utilisent une base courte (bonne base) du réseau euclidien qui doit rester secrète afin de garantir la sécurité du cryptosystème, mais ceux-ci laissent fuir des informations sur cette base secrète à chaque utilisation ce qui a permis des attaques dévastatrices [NR06; DN12b].

La méthode GPV corrige ce problème en modifiant la distribution de sortie de l'algorithme de décodage afin de la rendre indépendante de la base secrète utilisée. Elle permet ainsi d'obtenir des schémas de signature profitant de fortes garanties de sécurité grâce à l'utilisation d'algorithmes de décodage randomisés selon une loi gaussienne discrète dont le centre est dépendant du message à signer et n'est donc pas connu à priori. Cependant, les algorithmes d'échantillonnage selon ces distributions gaussiennes discrètes répondant aux fortes contraintes de la cryptographie sont encore insatisfaisant.

D'un côté, les techniques d'échantillonnage dépendantes du centre de la distribution, comme la méthode par inversion (par tables d'inverses), la méthode Knuth-Yao, l'alias méthode, la méthode *Zigurat* discrète et leurs variantes, sont les plus rapides connues pour échantillonner selon une loi gaussienne discrète. Cependant, elles utilisent des tables de pré-calculs relativement grandes pour chaque centre réel possible dans $[0, 1[$, ce qui les rend inefficaces quand le centre de la distribution est variable. De l'autre côté, les méthodes par rejet permettent d'échantillonner selon une loi gaussienne discrète pour tout centre réel sans pré-calculs importants, mais nécessitent des calculs coûteux et plusieurs essais par échantillon.

Dans cet article, nous nous intéressons à la réduction du nombre de centres pour lesquels nous avons à pré-calculer des tables lorsque l'on veut utiliser des méthodes dépendantes du centre pour échantillonner selon un centre variable et nous proposons un algorithme d'échantillonnage en centre variable : *Twin-CDT*, aussi rapide que sa variante en centre fixe. Enfin, nous présentons des résultats expérimentaux provenant de notre implémentation open-source en C++ qui indiquent que notre algorithme d'échantillonnage améliore le débit de l'algorithme de décodage randomisé de Peikert par un facteur 3 avec au plus 6,2 Mo de mémoire dédiée.

Nous considérons le cas où le centre de la Gaussienne n'est pas connu avant l'échantillonnage, comme c'est le cas en signature hache-puis-signe sur les réseaux euclidiens (comme Falcon). Le centre peut-être n'importe quel nombre réel, mais sans perte de généralité on peut considérer uniquement les centres dans $[0, 1)$. Parce que les tables d'inverses sont dépendantes du centre, une première option naïve serait de pré-calculer une table d'inverses pour chaque centre possible dans $[0, 1)$ selon la précision souhaitée. Évidemment, cette première option a le même coût calculatoire que l'algorithme en centre fixe utilisant une unique table d'inverses, i.e. $\mathcal{O}(\lambda \log s\lambda)$, pour λ le paramètre de sécurité et s le paramètre de la gaussienne. Cependant, cette option est complètement impraticable avec 2^λ tables d'inverses pré-calculées de taille $\mathcal{O}(s\lambda^{1.5})$. Un compromis opposé consiste à calculer la table d'inverses à-la-volée, évitant tout coût de stockage, qui augmente le coût calculatoire à un $\mathcal{O}(s\lambda^{3.5})$ en supposant que le coût de calcul de la fonction exponentielle est en $\mathcal{O}(\lambda^3)$.

Un nouvel algorithme d'échantillonnage gaussien en centre arbitraire. Une question intéressante consiste à se demander si nous pouvons garder le même coût calculatoire que l'algorithme en centre fixe avec un nombre polynomial de nombre de tables d'inverses. Pour répondre à cette question, nous commençons par fixer le nombre n de centres également espacés dans $[0, 1)$ et pré-calculons les tables d'inverses pour ces centres. Ensuite, nous appliquons l'algorithme par table d'inverse aux deux tables pré-calculés, pour lesquels les centres sont les plus proches du centre

souhaité, avec la même probabilité cumulée tirée uniformément. En supposant que le nombre de centre soit suffisant, les valeurs retournées par les deux tables d'inverses seront la plupart du temps les mêmes, dans ce cas nous pouvons conclure, grâce à un simple argument de monotonie, que la valeur retournée aurait été la même pour la table d'inverse calculée du centre souhaité. Nous pouvons ainsi retourner cette valeur comme un échantillon valide. Sinon, la plus grande valeur va immédiatement suivre la plus petite et nous devons calculer la fonction de répartition de la petite valeur pour le centre souhaité afin de savoir si la probabilité cumulée tirée uniformément est plus petite ou plus grande. Si elle est plus plus petite, la plus petite valeur est renvoyée, sinon c'est la plus grande.

Comme dit précédemment, pour réduire la mémoire nécessaire à l'exécution de l'algorithme par table d'inverses quand le centre est déterminé durant l'échantillonnage, nous pouvons pré-calculer les tables d'inverses pour un nombre n de centres également espacés dans $[0, 1)$ et calculer la fonction de répartition quand nécessaire. L'algorithme 0.1, respectivement 0.2, décrivent les phases de pré-calculs, resp. d'échantillonnage, de l'algorithme *Twin-CDT*. L'algorithme 0.1 pré-calcule les

Algorithm 0.1 Twin-CDT : Pré-calculs

Entrée : un paramètre gaussien s et un nombre de contres n
Sortie : une matrice pré-calculée $\mathbf{T}$

initialiser une matrice vide $\mathbf{T} \in \mathbb{FP}_{\lambda}^{n \times 2\lceil \tau s \rceil + 3}$
for $i \leftarrow 0, \ldots, n-1$ **do**
    for $j \leftarrow 0, \ldots, 2\lceil \tau s \rceil + 2$ **do**
        $\mathbf{T}_{i,j} \leftarrow \mathbb{FP}_m : \mathrm{cdf}_{s,i/n}(j - \lceil \tau s \rceil - 1)$

tables d'inverses, jusqu'à une précision de mantisse m qui garantit que les λ bits les plus significatifs soient corrects pour chaque évaluation de la fonction de répartition, puis les stocke avec λ bits de précision comme une matrice $\mathbf{T}$, où la i-ème ligne est la table d'inverses correspondante au i-ème centre pré-calculé i/n.

Algorithm 0.2 Twin-CDT : Échantillonnage

Entrée : un centre c et une matrice pré-calculée $\mathbf{T}$
Sortie : un échantillon x que suit $D_{s,c}$

$p \leftarrow 0.U_{\{0,1\}^{\lambda}}$
$v_1 \leftarrow i - \lceil \tau s \rceil - 1$ t.q. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i}$
$v_2 \leftarrow j - \lceil \tau s \rceil - 1$ t.q. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor) \rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor) \rceil, j}$
if $v_1 = v_2$ **then**

    return $v_1 + \lfloor c \rfloor$
else
    if $p < \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**

        return $v_1 + \lfloor c \rfloor$
    else

        return $v_2 + \lfloor c \rfloor$

Pour échantillonner selon la distribution gaussienne discrète $D_{\mathbb{Z},s,c}$, l' algorithme 0.2 cherche un

antécédent par la fonction de répartition d'une probabilité cumulée p, tirée suivant une distribution uniforme sur $[0, 1)$, dans les deux tables d'inverses correspondantes au centre $\lfloor n(c - \lfloor c \rfloor) \rfloor / n$ (respectivement $\lceil n(c - \lfloor c \rfloor) \rceil / n$) qui retourne uns valeur v_1 (resp. v_2). Si la même valeur est retournée pour les deux tables d'inverses (i.e. $v_1 = v_2$), alors cette valeur additionnée à la partie entière du centre souhaité est un échantillon valide, sinon on calcule $\mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ et retourne $v_1 + \lfloor c \rfloor$ si $p < \mathrm{cdf}_{s,c}(v_1)$ et $v_2 + \lfloor c \rfloor$ sinon.

Échantillonnage gaussien par tables d'inverses : de la multi à la double précision

Cet article intitulé *CDT-based Gaussian Sampling : From Multi to Double Precision* est le fruit d'une collaboration avec Carlos Aguilar-Melchor et a été publié dans le journal *IEEE Transactions on Computers*.

Afin de garantir la sécurité des schémas cryptographiques les employant, il est nécessaire de limiter l'écart entre la distribution de probabilité en sortie du générateur gaussien et la distribution théorique. Pour ce faire, nous utilisons généralement la distance statistique en cryptographie. La divergence de Rényi est une seconde méthode permettant de mesurer l'écart entre deux distributions de probabilités qui a trouvé plusieurs applications en cryptographie à base de réseaux euclidiens durant ces dernières années comme alternative à la distance statistique. En particulier une borne intéressante a été présentée récemment pour la divergence de Rényi de distributions ayant une erreur relative bornée.

Dans cet article nous montrons que cette borne peut être utilisée pour limiter la précision requise de certains algorithmes d'échantillonnage gaussien utilisés en cryptographie à la double précision standard en point-flottant définie par la norme IEEE 754 pour les paramètres usuels en signature basée sur les réseaux euclidiens. Cette technique, reposant sur une modification simple de la table de pré-calculs de la fonction de répartition, réduit la mémoire utilisée par ces algorithmes et rend leur implémentation en temps-constant plus simple et plus rapide.

Nous appliquons également cette technique à notre algorithme Twin-CDT d'échantillonnage en centre variable qui nécessite occasionnellement une évaluation de la fonction de répartition. Ainsi, la quantité de calculs coûteux en point-flottant est drastiquement réduite rendant l'implémentation temps-contant et résistante au attaques par cache de cet algorithme viable et efficace. Enfin, nous présentons des résultats expérimentaux qui indiquent qu'en comparaison avec une méthode par rejet, notre approche est jusqu'à 75 fois plus rapide que la plus rapide des méthodes existantes en centre variable (l'échantillonneur de Karney) et améliore le débit de signature de GPV par un facteur 4 à 8 en fonction du paramètre de sécurité.

L'arithmétique en virgule-flottante. Les ordinateurs ayant une mémoire limitée, ils ne peuvent donc pas manipuler avec exactitude des nombres réels. En pratique, on peut utiliser l'arithmétique en virgule-flottante (FPA) pour manipuler des nombres réels approximés. L'arithmétique en virgule-flottante est une arithmétique utilisant une représentation sous forme de formule des nombres réels comme une approximation permettant différents compromis entre domaine et précision. Un nombre réel est représenté approximativementen FPA, suivant le format binaire en double précision IEEE 754 (binary64), comme une mantisse $m \in]-2, -1] \cup [1, 2[$ de précision $p = 53$ bits tel que $|m| \cdot 2^{52} \in [2^{52}, 2^{53} - 1] \cap \mathbb{Z}$, décalé grâce à un exposant (biaisé) de 11 bits $e \in [-1022, 1023] \cap \mathbb{Z}$ en base deux. Tel que tout nombre en virgule-flottante $\bar{x} \in \mathbb{FP}_p$ (i.e. tout nombre qui peut être représenter exactement comme un nombre en virgule-flottante étant donnée une précision p) est de la forme $\bar{x} = m \cdot 2^e$. Il est intéressant de noter que, dans le format IEEE 754 binary64, le bit le

plus significatif de $|m|$ est toujours égal à un, il n'a donc pas besoin d'être stocké, la mantisse est représentée par un bit de signe et les 52 bits de sa partie décimale.

En arithmétique en virgule-flottante, soit $\bar{x}$ l'approximation en virgule-flottante avec p bits du mantisse d'un nombre réel x, l'erreur relative de $\bar{x}$ est définie comme $\delta_{\mathsf{RE}}(x, \bar{x}) := |x - \bar{x}|/|x|$. La norme IEEE 754 sur les nombres en virgule-flottante garantit que l'erreur relative est bornée par $\delta_{\mathsf{RE}}(x, \bar{x}) \leq 2^{1-p}$ pour toute opération arithmétique élémentaire (addition, soustraction, multiplication, division, racine carrée et multiplication-avec-accumulation) et les implémentations usuelles calculent aussi les fonctions transcendantales basiques avec $\delta_{\mathsf{RE}}(x, \bar{x}) \leq 2^{2-p}$. Rappelons aussi la notion d'erreur relative de Micciancio et Walter étendue à toutes distributions P et Q avec le même support $S := \mathrm{Supp}(P) = \mathrm{Supp}(Q)$:

$$\delta_{\mathsf{RE}}(P, Q) := \max_{x \in S} \delta_{\mathsf{RE}}(P(x), Q(x)) = \max_{x \in S} \frac{|P(x) - Q(x)|}{P(x)}.$$

Remarquons également que la distance statistique est bornée par $\Delta(P, Q) \leq \frac{1}{2}\delta_{\mathsf{RE}}(P, Q)$. Et que différentes bornes ont successivement été présentées pour la divergence de Rényi de deux distributions P et Q de même support dont l'erreur relative est bornée. Pour $\delta_{\mathsf{RE}}(P, Q) \leq 1/4$, on a :

$$R_1(P\|Q) \leq \exp\left(\delta_{\mathsf{RE}}(P, Q)^2\right) \text{ ([PDG14, Lemme 2] renforcé dans [MW17, Lemme 2.1]).} \quad (1)$$

Et pour $a > 1$:

$$R_a(P\|Q) \leq \left(1 + \frac{a(a-1)\delta_{\mathsf{RE}}(P, Q)^2}{2(1 - \delta_{\mathsf{RE}}(P, Q))^{a+1}}\right)^{\frac{1}{a-1}} \text{ ([Pre17, Lemme 3]).} \quad (2)$$

Les mesures de proximité entre deux distributions. Si nous supposons que nos ordinateurs peuvent manipuler exactement des nombres réels, nous pouvons demander aux séquences aléatoires obtenues en sortie d'un générateur de nombres non-uniformes de suivre exactement la distribution souhaitée. Les algorithmes ou générateurs avec cette propriété sont appelés exactes. Des générateurs exacts sont réalisables, pour des densités non-transcendantales, si nous utilisons une arithmétique en précision étendue. Cependant, pour des raisons d'efficacité, dans ce manuscrit nous nous intéressons aux générateur inexacts, qui sont généralement des algorithmes basés sur des approximations mathématiques. Cependant, en cryptographie il est préférable d'avoir des garanties de sécurité, nous utilisons donc une mesure de proximité afin de déterminer la précision requise pour la mise-en-œuvre de générateurs de nombres aléatoires non-uniformes.

La distance statistique et la divergence de Rényi sont deux mesures de proximité entre deux distributions de probabilité. La divergence de Rényi a trouvé plusieurs applications ces dernières années en cryptographie basée sur les réseaux euclidiens comme alternative à la distance statistique. La distance statistique entre deux distributions discrètes P et Q, avec le même supporte $S := \mathrm{Supp}(P) = \mathrm{Supp}(Q)$, est définie comme :

$$\Delta(P, Q) := \frac{1}{2}\sum_{x \in S} |P(x) - Q(x)|$$

La divergence de Rényi est une alternative à la distance statistique, où la différence de la distance statistique est remplacée par un ratio dans la divergence de Rényi. Pour toutes distributions P et Q telles que $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$ et $a \in]1, +\infty[$, on définie la divergence de Rényi d'ordre a par :

$$R_a(P\|Q) := \left(\sum_{x \in \mathrm{Supp}(P)} \frac{P(x)^a}{Q(x)^{a-1}}\right)^{\frac{1}{a-1}}$$

On définie aussi la divergence de Rényi d'ordres 1 et $+\infty$ par : We define the Rényi divergences of orders 1 and $+\infty$ by

$$R_1(P\|Q) := \exp\left(\sum_{x\in \mathrm{Supp}(P)} P(x)\log\frac{P(x)}{Q(x)}\right)$$

et

$$R_\infty(P\|Q) := \max_{x\in\mathrm{Supp}(P)} \frac{P(x)}{Q(x)}$$

Notons que la divergence de Rényi d'ordre 1 est connue comme (l'exponentielle de) la divergence de Kullback-Leibler et remarquons que la divergence de Rényi n'est pas une distance.

Nous rappelons également quelques propriétés importantes de la divergence de Rényi de [BLL+15; EH14] : Soit $a \in [1, +\infty]$. Soit P et Q deux distributions avec $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$. On a les propriétés suivantes :

- Pour toute fonction f, où P^f (resp. Q^f) dénote la distribution de $f(y)$ obtenue en tirant y selon P (resp. Q), on a :
$$R_a(P^f\|Q^f) \leq R_a(P\|Q).$$
- Supposons que P et Q soient les deux distributions d'une pair de variables aléatoires (Y_1, Y_2). Pour $i \in \{1, 2\}$, soit P_i (resp. Q_i) la distribution de Y_i sous P (resp. Q). Alors, si Y_1 et Y_2 sont indépendantes, on a :
$$R_a(P\|Q) = R_a(P_1\|Q_1)R_a(P_2\|Q_2).$$
- Soit $A \subseteq \mathrm{Supp}(Q)$ un événement arbitraire. Si $a \in]1, +\infty[$, alors :
$$Q(A) \geq P(A)^{\frac{a}{a-1}}/R_a(P\|Q),$$
$$Q(A) \geq P(A)/R_\infty(P\|Q).$$

Une question importante, à propos de l'arithmétique en virgule-flottante utilisée dans un algorithme cryptographique, est de savoir comment celle-ci affecte les performances et la sécurité de cet algorithme. En effet, comme expliqué dans [DN12a] la précision utilisée pour l'arithmétique en virgule-flottante a un impacte important, car les opérations en virgule-flottante deviennent bien plus coûteuses lorsque la précision dépasse la précision matérielle supportée dans l'unité arithmétique et logique du processeur. En particulier, les processeurs modernes fournissent une arithmétique en virgule-flottante suivant la double précision standard IEEE 754 ($p = 53$), mais la quadruple précision ($p = 113$) est généralement environ 10 à 20 fois plus lente pour les opérations basiques.

L'algorithme CDT. Rappelons que la méthode d'échantillonnage par inversion repose sur l'observation que si X est une variable aléatoire continue de fonction de répartition cdf, alors $\mathrm{cdf}(X)$ suit une distribution uniforme sur $[0, 1[$. D'où la méthode par inversion : $\mathrm{cdf}^{-1}(U_{[0,1)})$ a la même distribution que X. L'algorithme CDT [Pei10] est basé sur cette méthode avec la spécificité que les images par la fonction de répartition sont pré-calculées et stockées dans une table, nommée CDT, pour le sous-ensemble significatif du domaine. Par sous-ensemble signification du domaine nous entendons un intervalle suffisamment large en accord avec la distance statistique souhaitée. En effet, en supposant que les valeurs de la table CDT pré-calculée soit stockée avec une précision infinie, la distance statistique entre la sortie de l'algorithme CDT et la distribution parfaite est égale à la moitié de la somme des probabilités non-couvertes par la table pré-calculée.

Algorithm 0.3 CDT : Pré-calculs

Entrée : un paramètre gaussien s et un centre c
Sortie : une table pré-calculée $\mathbf{T}$

1: initialiser une table vide $\mathbf{T}$
2: **for** $i \leftarrow 0, \ldots, 2\lceil ts \rceil$ **do**
3: $\quad \mathbf{T}_i \leftarrow \mathrm{cdf}_{s,c}(i - \lceil ts \rceil)$

Algorithm 0.4 CDT : Échantillonnage

Entrée : une table pré-calculée $\mathbf{T}$
Sortie : un échantillon x que suit $D_{s,c}$

1: $u \leftarrow U_{[0,1)}$
2:
3: **return** $i - \lceil ts \rceil$ t.q. $\mathbf{T}_{i-1} \leq u < \mathbf{T}_i$

Réduire l'erreur relative de l'algorithme CDT. Soit $\bar{D}_{s,c}$ la distribution de sortie de l'algorithme 0.4, soit $S := [-\lceil ts \rceil, \lceil ts \rceil] \cap \mathbb{Z}$ le support tronqué et p la précision de la mantisse, on a :

$$\delta_{\mathsf{RE}}(D_{s,c}, \bar{D}_{s,c}) \leq \frac{\rho_{s,c}(\mathbb{Z}) - \rho_{s,c}(S)}{2} + \max_{x \in S} \frac{\mathrm{cdf}_{s,c}(x)}{D_{s,c}(x)} 2^{-p}$$

En considérant une CDT dans l'ordre naturel, comme générée par l'algorithme 0.3, l'erreur relative de $\bar{D}_{s,c}$ est significativement large dû au fait que la queue de $D_{s,c}(x)$ devient très faible lorsque $\mathrm{cdf}_{s,c}(x) \approx 1$. Une solution pour réduire cette erreur relative consiste à réordonner le support S tel que les plus faibles probabilités soient avant les plus grandes dans la table CDT. Pour ce faire, nous employons une relation d'ordre total $\prec$ sur le support S, définie pour tout entiers x et y comme :

$$x \prec y \text{ si et seulement si } \begin{cases} D_{s,c}(x) < D_{s,c}(y) \\ D_{s,c}(x) = D_{s,c}(y) \text{ et } x < y \end{cases}.$$

Nous notons aussi $\preceq$ la relation qui étend $\prec$ en ajoutant que x est en relation avec lui-même, i.e. $x \preceq y$ si et seulement si, $x \prec y$ ou $x = y$.

On appelle $S^{\prec} := (x_i)_{0 \leq i \leq 2\lceil ts \rceil} := (S, \prec)$ le support réordonné en accord avec la relation d'ordre $\prec$. En supposant que $c \in [0, 1[$, la permutation pour passer de S à $S^{\prec}$ est plutôt simple :

$$\begin{aligned} S^{\prec} &= ((-1)^{i+1}(\lceil ts \rceil - i))_{0 \leq i \leq 2\lceil ts \rceil - 2} \| (b_0, b_1) \\ &= (-\lceil ts \rceil, \lceil ts \rceil, -\lceil ts \rceil + 1, \lceil ts \rceil - 1, \ldots, -2, 2, -1, b_0, b_1) \end{aligned}$$

où

$$(b_0, b_1) = \begin{cases} (1, 0) & \text{si } 0 \leq c < \frac{1}{2} \\ (0, 1) & \text{si } \frac{1}{2} \leq c < 1 \end{cases}$$

Maintenant nous pouvons définir une nouvelle fonction de répartition de $D_{s,c}$ sur $S^{\prec}$, pour tout entier $x \in S$:

$$\mathrm{rcdf}_{s,c}(x) := \sum_{y \in S}^{y \preceq x} D_{s,c}(y).$$

Algorithm 0.5 CDT réordonnée : Pré-calcul

Entrée : un paramètre gaussien s, un centre c et un support réordonné tronqué $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Sortie : une CDT réordonnée pré-calculée $\mathbf{T}$

1: initialiser une table vide $\mathbf{T}$
2: **for** $i \leftarrow 0, \ldots, 2\lceil ts \rceil$ **do**
3: $\quad \mathbf{T}_i \leftarrow \mathrm{rcdf}_{s,c}(x_i)$

Algorithm 0.6 CDT réordonnée : Échantillonnage

Entrée : une CDT réordonnée pré-calculée $\mathbf{T}$ et un support réordonné tronqué $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Sortie : un échantillon x suivant $D_{s,c}$

1: $u \leftarrow U_{[0,1]}$
2:
3: **return** x_i t.q. $\mathbf{T}_{i-1} \leq u < \mathbf{T}_i$

Nous modifions les algorithmes 0.3 et 0.4 en les algorithmes 0.5 et 0.6 pour utiliser rcdf au lieu de cdf.

Soit $\bar{D}^{\prec}_{s,c}$ la distribution de sortie de l'algorithme 0.6 et p la précision de la mantisse, pour $s \geq 1$, cette CDT réordonnée (rCDT) réduit l'erreur relative des probabilités cumulées stockées à :

$$\delta_{\mathsf{RE}}(D_{s,c}, \bar{D}^{\prec}_{s,c}) < 2^{-p+2.3+\log_2 s} \tag{3}$$

Démonstration. Soit S^-, S^+ respectivement les parts négative et strictement positives de S, i.e. $S^- := [-\lceil ts \rceil, 0] \cap \mathbb{Z}$ et $S^+ := [1, \lceil ts \rceil] \cap \mathbb{Z}$. On sait que :

$$\frac{\mathrm{rcdf}_{s,c}(x)}{D_{s,c}(x)} = \sum_{y \in S^-}^{y \preceq x} \frac{D_{s,c}(y)}{D_{s,c}(x)} + \sum_{z \in S^+}^{z \preceq x} \frac{D_{s,c}(z)}{D_{s,c}(x)}$$

Et, par une comparaison somme-intégrale :

$$\begin{aligned} \sum_{y \in S^-}^{y \preceq x} \frac{D_{s,c}(y)}{D_{s,c}(x)} + \sum_{z \in S^+}^{z \preceq x} \frac{D_{s,c}(z)}{D_{s,c}(x)} &\leq 2 + 2 \int_{y=|x|}^{\infty} \frac{\rho_{s,c}(y)}{\rho_{s,c}(|x|)} \\ &\leq 2 + s\sqrt{2\pi} \end{aligned}$$

□

Dans des travaux précédents, la précision été déterminée par des analyses basées sur la distance statistique [Pei10] ou sur la divergence de Kullback-Leibler [PDG14] pour une CDT classique et une demie-CDT renversée. Dans la suite, les distributions Φ et Φ' représentent le schéma cryptographique vue par l'adversaire dans le cas approximé (resp. idéal). Nous supposons qu'une requête à l'oracle de fonctionnalité clé-privée correspond à m requêtes à l'algorithme d'échantillonnage gaussien, avec un paramètre gaussien maximum $s := \max_{i=1,\ldots,m} s_i$, et, en accord avec l'appel à proposition du NIST pour une standardisation post-quantique, nous limitions le nombre de requêtes de l'attaquant à l'oracle de fonctionnalité clé-privée par $q_s = 2^{64}$.

Analyse basée sur la distance statistique. Tout adversaire avec une probabilité de succès ε' sur le schéma implémenté avec un échantillonnage gaussien parfait a une probabilité $\varepsilon \leq \varepsilon' + \Delta(\Phi, \Phi')$ contre le schéma implémenté avec un échantillonnage approché. Supposons que les paramètres pour le schéma idéal sont sélectionnés pour avoir $\varepsilon' \leq 2^{-\lambda-1}$. Pour assurer une sécurité contre mq_s requêtes, chaque variable aléatoire gaussienne approchée $(\bar{D}_{s_i,c_i})_i$ doit être à une distance statistique $\Delta(\Phi, \Phi')/(mq_s)$ de la distribution désirée $(D_{s_i,c_i})_i$. En utilisant 3, on a $\Delta(\bar{D}_{s_i,c_i}, D_{s_i,c_i}) \leq 2^{-p+1.3+\log_2 s_i}$, ce qui nous amène à une précision de mantisse requise dans la CDT réordonnée pour λ bits de sécurité :

$$p \geq \lambda + 2.3 + \log_2(smq_s).$$

Analyse basée sur la divergence de Kullback-Leibler. Dans [PDG14] la distance statistique est remplacée par la divergence de Kullback-Leibler, i.e. la divergence de Rényi divergence d'ordre $a = 1$, pour réduire la précision p de la table pré-calculée. Ils montrent que pour tout adversaire avec une probabilité de succès $\varepsilon' \leq 2^{-\lambda-1}$ sur le schéma implémenté avec un échantillonnage gaussien parfait a une probabilité de succès $\varepsilon \leq \varepsilon' + \sqrt{\log R_1(\Phi\|\Phi')/2}$ contre le schéma implémenté avec un échantillonnage gaussien approché. Supposons que les paramètres pour le schéma idéal sont sélectionnés pour avoir $\varepsilon' \leq 2^{-\lambda-1}$. Par la propriété multiplicative de la Rényi divergence sur mq_s échantillons indépendants, on a $R_1(\Phi\|\Phi') \leq (\max_{i=1,\ldots,m} R_1(\bar{D}_{s_i,c_i}\|D_{s_i,c_i}))^{mq_s}$. En utilisant 3 et 1 nous amène à une précision de mantisse requise dans la CDT réordonnée pour λ bits de sécurité :

$$p \geq \lambda + 3.3 + \log_2(s\sqrt{mq_s}).$$

Analyse basée sur la divergence de Rényi. Comme décrit dans [BLL+15], la propriété de conservation des probabilité de la divergence de Rényi est multiplicative pour $a > 1$ au lieu d'additive pour $a = 1$ et pour la distance statistique. Tout adversaire avec une probabilité de succès ε' sur le schéma implémenté avec un échantillonnage gaussien parfait a une probabilité $\varepsilon \leq (\varepsilon' R_a(\Phi\|\Phi'))^{\frac{a-1}{a}}$ contre le schéma implémenté avec un échantillonnage approché. Supposons que les paramètres pour le schéma idéal sont sélectionnés pour avoir $\varepsilon' \leq 2^{-\frac{a}{a-1}\lambda-1}$, i.e. supposons que pour tout $k > 0$ on ait $\lambda \leq (a-1)k$, alors les paramètres pour le schéma idéal sont sélectionnés pour avoir $(\lambda + k + 1)$ bits de sécurité. Par la propriété multiplicative de la Rényi divergence sur les mq_s échantillons indépendants, on a $R_a(\Phi\|\Phi') \leq (\max_{i=1,\ldots,m} R_a(\bar{D}_{s_i,c_i}\|D_{s_i,c_i}))^{mq_s}$. En utilisant 3 et 2 nous amène à une précision de mantisse requise dans la CDT réordonnée pour λ bits de sécurité :

$$p \geq 1.3 + \log_2(s\sqrt{amq_s})$$

En particulier, si on prend $a = 256$, on peut prendre $p = 53$, i.e. la précision du standard double précision IEEE 754, en respectant $\log_2(s\sqrt{mq_s}) \leq 47.7$, ce qui est habituellement le cas pour les schémas de signature basés sur les réseaux euclidiens où $q_s = 2^{64}$, $m \leq 2^{11}$ et $s \leq 2^8$.

En comparant les analyses de sécurité ci-dessus, on remarque que la divergence de Rényi permet d'obtenir la précision de mantisse la plus faible. De plus, cette taille est, dans l'analyse de sécurité basée sur la divergence de Rényi, est indépendante du paramètre de sécurité ciblé, ce qui est particulièrement intéressant. Avec $p = 53$, i.e. la précision du standard double précision IEEE 754, cela nous permet de limité la taille de la table CDT complète pré-calculée à $128\lceil s\sqrt{\lambda}/2\rceil + 64$ bits et une demie table CDT pré-calculée à $64\lceil s\sqrt{\lambda}/2\rceil + 64$ bits. En effet, quant la distribution gaussienne est symétrique, une demie table de probabilités suffit. La méthode d'échantillonnage consiste alors à

échantillonner selon la demie distribution, avec une probabilité deux fois plus petite pour le centre de la symétrie, avant de tirer un bit uniforme pour déterminer le signe.

Outre la réduction de mémoire utilisée, l'utilisation de la double précision standard IEEE 754 permet d'accélérer significativement les calculs dans le cas d'un échantillonnage en centre variable, comme utilisé dans les schémas de signature hache-puis-signe basés sur les réseaux euclidiens. Elle permet aussi de stocker chaque entrée de la table dans un seul registre ce qui permet une protection plus simple et efficace de l'algorithme d'échantillonnage contre les attaques par observation du temps d'exécution ou par manipulation du cache.

Déléguer des opérations cryptographiques avec le chiffrement homomorphe

Cet article intitulé *Delegating Elliptic-Curve Operations with Homomorphic Encryption* est le fruit d'une collaboration avec Carlos Aguilar-Melchor, Jean-Christophe Deneuville, Philippe Gaborit et Tancrède Lepoint et a été présenté au séminaire *IEEE Workshop on Security and Privacy in the Cloud* (SPC) 2018.

Le paysage du chiffrement entièrement homomorphe (FHE/SWHE) a connu de grands changements au cours des six dernières années. À mesure que les coûts de calcul chutent, des bibliothèques implémentant ces schémas sont développées et de nouvelles applications deviennent possibles. Plusieurs prototypes, démontrant leur utilité concernant le diagnostique médical, le traitement du signal, les statistiques génomiques et l'accès confidentiel à des bases de données, ont suscité un vif espoir quant au déploiement du chiffrement entièrement homomorphe dans un proche avenir. Cependant, dans la plupart de ces applications, augmenter la sécurité et les fonctionnalités amène d'important coûts en temps et en communications.

Dans cet article, nous nous éloignons de l'utilisation bit-à-bit du chiffrement entièrement homomorphe et démontrons qu'appliqué à l'évaluation de circuits arithmétiques sur des données chiffrées, les schémas de chiffrement entièrement homomorphe peuvent être efficaces en pratique. En particulier, nous nous intéressons à la délégation d'opérations pour la cryptographie basée sur les courbes elliptiques. Plus précisément, nous montrons comment réduire la charge de calcul de la multiplication scalaire du générateur par un scalaire secret. Nous montrons qu'il est ainsi possible de réduire ces coûts de calcul, même par rapport aux protocoles de délégation traditionnels.

L'efficacité pratique de notre protocole est démontrée par une implémentation utilisant notre adaptation de la bibliothèque HElib, modifiée afin de pouvoir effectuer des calculs dans un espace des clairs avec un module multi-précision.

References

[BLL+15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. *Improved Security Proofs in Lattice-Based Cryptography: Using the Rényi Divergence Rather Than the Statistical Distance*. In: *ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. Springer, Heidelberg, Nov. 2015, pp. 3–24 (cit. on pp. xix, xxii, 29, 30, 41, 44, 93).

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. *Efficient Identity-Based Encryption over NTRU Lattices*. In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 22–41 (cit. on pp. xiii, 54, 62, 63, 65, 66, 70).

[DN12a] Léo Ducas and Phong Q. Nguyen. *Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 415–432 (cit. on pp. xix, 32, 33, 36, 37, 40).

[DN12b] Léo Ducas and Phong Q. Nguyen. *Learning a Zonotope and More: Cryptanalysis of NTRUSign Countermeasures*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 433–450 (cit. on pp. xiii, xv, 8, 62, 64, 68).

[DP16] Léo Ducas and Thomas Prest. *Fast Fourier Orthogonalization*. In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 191–198. ISBN: 978-1-4503-4380-0. URL: http://doi.acm.org/10.1145/2930889.2930923 (cit. on pp. xiii, xiv, 62, 67).

[EH14] Tim van Erven and Peter Harremoës. "Rényi Divergence and Kullback-Leibler Divergence". In: *IEEE Trans. Information Theory* 60.7 (2014), pp. 3797–3820 (cit. on pp. xix, 30).

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. *Public-Key Cryptosystems from Lattice Reduction Problems*. In: *CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 112–131 (cit. on pp. xiii, 7–9, 13, 62, 64).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. *Trapdoors for hard lattices and new cryptographic constructions*. In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206 (cit. on pp. xii, xiii, 9, 13, 14, 31, 62–65).

[HHP+03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. *NTRUSIGN: Digital Signatures Using the NTRU Lattice*. In: *CT-RSA 2003*. Ed. by Marc Joye. Vol. 2612. LNCS. Springer, Heidelberg, Apr. 2003, pp. 122–140 (cit. on pp. xiii, 8, 62, 64).

[Kle00] Philip N. Klein. *Finding the closest lattice vector when it's unusually close*. In: *11th SODA*. Ed. by David B. Shmoys. ACM-SIAM, Jan. 2000, pp. 937–941 (cit. on pp. xiv, 13, 63, 64, 67).

[MP12] Daniele Micciancio and Chris Peikert. *Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 700–718 (cit. on pp. xiv, 67).

[MW17] Daniele Micciancio and Michael Walter. *Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time*. In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 455–485 (cit. on pp. xviii, 27, 29, 30, 33, 69, 71).

[NR06] Phong Q. Nguyen and Oded Regev. *Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 271–288 (cit. on pp. xiii, xv, 7, 8, 13, 62, 64, 68).

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. *Enhanced Lattice-Based Signatures on Reconfigurable Hardware*. In: *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. Ed. by Lejla Batina and Matthew Robshaw. Springer Berlin Heidelberg, 2014, pp. 353–370 (cit. on pp. xviii, xxi, xxii, 30, 40, 43, 44).

[Pei10] Chris Peikert. *An Efficient and Parallel Gaussian Sampler for Lattices*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 80–97 (cit. on pp. xiv, xix, xxi, 14, 27, 33, 40, 41, 43, 53, 63, 67).

[Pre17] Thomas Prest. *Sharper Bounds in Lattice-Based Cryptography Using the Rényi Divergence*. In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Heidelberg, Dec. 2017, pp. 347–374 (cit. on pp. xviii, 30, 40, 68, 69, 92).

[SS11] Damien Stehlé and Ron Steinfeld. *Making NTRU as Secure as Worst-Case Problems over Ideal Lattices*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 27–47 (cit. on pp. xiii, 7, 62, 65, 66).

Contents

Every kind of science, if it has only reached a certain degree of maturity, automatically becomes a part of mathematics.

Axiomatic Thought – David Hilbert

1 Introduction

This thesis is about one small field of cryptography on the crossroads of number theory, algebraic geometry and computational complexity theory. Historically, cryptography was mainly about securing communications, in particular to ensure data confidentiality, data integrity and authentication. Yet over the past few decades, the field has blossomed into a discipline having much broader and richer goals, encompassing almost any scenario involving communication or computation in the presence of potentially malicious behavior.

Classical public-key cryptography is mainly built upon two mathematical problems which are assumed hard to solve with actual technology: the problem of factoring integers (FACT) and the discrete logarithm problem (DLOG). However, a technological innovation threats cryptographic algorithms relying on these problems: the "quantum computer". Indeed, *basic operations* of quantum and classical computers are different in nature making a quantum computer with enough qubits ables to efficiently solve FACT and DLOG. This threat motivated the design of new public-key cryptographic problems able to resist against quantum computers.

A new cryptography, based on geometrical problems over mathematical objects called lattices, is one of the most promising alternatives. Indeed, lattice-based cryptography has generated considerable interest in the last decade due to many attractive features, including conjectured security against quantum attacks, strong security guarantees from *worst-case hardness* assumptions and constructions of fully homomorphic encryption schemes. In this introduction we briefly survey some of the pioneering and modern works in lattice cryptography. Note that many parts of this introduction are extracted from the Peikert's survey [Pei16] and some from Micciancio's historical talk at the University of California, Berkeley, on July 6th 2015.

Contents

1.1. A New Promising Cryptography

A lattice is a set of points in space that are arranged regularly. One relevant feature of lattices is that these points are in a real space of dimension n, therefore lattices are mathematical objects that mix together some continuous and discrete properties. They are described by real numbers, but it is a discrete set of points in the topology sense that these points are far apart from each other. We will see how these can map in the study of lattice cryptography.

Any history about lattices in the context of computer science has necessarily to start with the Lenstra-Lenstra-Lovász (LLL) algorithm presented in their 1982 paper [LLL82]. This paper is know for its algorithms that can be used to efficiently find approximate solutions to lattice problems. The approximation factor grows exponentially with the dimension of the lattice, but this algorithm works well in practice and it has been used in cryptanalysis to break some schemes, e.g. the Merkle-Hellman cryptosystem [Sha84].

However, it has been necessary to wait 1996 to have the idea to use lattices in the design of cryptographic primitives [Ajt96], which marks the beginning of this new lattice-based cryptography.

1.1.1. Strong Security Guarantees

Cryptography inherently requires average-case intractability, i.e. problems for which random instances (drawn from a specified probability distribution) are hard to solve. This is qualitatively different from the worst-case notion of hardness usually considered in the computational complexity theory, where a problem is considered hard if there merely exist some difficult instances. Problems that appear hard in the worst case often turn out to be easier on the average, especially for distributions that produce instances having some extra "structure", e.g. the existence of a secret key for decryption.

In his groundbreaking work, Ajtai [Ajt96] gave the first *worst-case to average-case reductions* for lattice problems, and with them the first cryptographic object with a proof of security assuming the hardness of well-studied computational problems on lattices. In particular, Ajtai's work gave the first cryptographic function based on a standard *worst-case* complexity assumption of any kind. Ajtai introduced the (average-case) "short integer solution" (SIS) problem and its associated one-way function, and proved that solving it is at least as hard as approximating various lattice problems in the worst case. Both SIS and Ajtai's function are still heavily used to this day.

Worst-case to average-case reduction. In order to find a "hard problem", one usually resorts to one of two possible methods. One is to choose an NP-hard problem from the computational complexity theory. NP-hard and NP-complete problems are as hard as any other problem in the same class.

If any problem among the NP-hard problems, e.g. the NP-hard problem we have chosen, can be solved efficiently then all the NP-hard problems can also be solved efficiently. Thus chosing an NP-hard problem ensures that there will not be an efficient solution for it except if a groundbreaking algorithm solves all the hard problems known at the same time, which does not seem plausible. Alternatively, one can choose a question that has been studied for a long time, such as prime factorization. Indeed, we believe that prime factoring is hard, at least classically, and there is evidence of hardness there.

However, regardless of the method applied, the problem is only assumed hard in the worst case. In other words, there is no algorithm that could solve efficiently every instances, but there may be some weak instances. And, as a consequence on cryptographic applications, there is no guidance about

how to create hard instances of the problem. A possible solution is to find a set of random instances of the problem, and then show that if we can solve these random instances with non-negligible probability, then we can solve the hard mathematical problem in the worst case. This allows us to pick instances according to a random distribution with the guarantee that these random instances are as hard as the hardest instance. That is exactly what Ajtai did in his paper of 1996.

Discrete-log cryptography versus lattice-based cryptography. Note that we already have a similar property with the discrete logarithm problem (DLOG) which is random self-reducible. Let G be any group, denote its group operation by multiplication and let g be a generator of G. Given g^h the DLOG problem consists in find h. This problem is random self-reducible in the sense that if we can solve this problem when g and h are chosen at random, then we can solve the problem also in the worst case. This means that we know how to choose g and h to make the problem hard. However, this does not give any guidance about how to choose the group G.

So what is the difference between these two kinds of hardness? In the lattice case, the assumption is that there is no algorithm that can solve lattice problems efficiently. Specifically, the complexity of solving lattice problems grow exponentially or at least superpolynomially in the dimension of the lattice n. In saying that, we consider a set of distributions, one for every value of n. In the same way, for the DLOG problem we consider a set of distributions, one for every group G, and the conjecture is that the complexity of solving DLOG grows superpolynomially in the bit size of the order of G.

However, these two problems are not quite the same. The reason is that, since to choose a lattice of dimension n at random guarantees to have a hard instance, n is a security parameter, while for a fixed bit size of the order we have exponentially many possible groups G without any guarantee on the hardness of their instance.

In the case of lattices, considering the sequence of problems of increasing dimension we can noticed that lattice problems in dimension n can be reduced to lattice problems in a larger dimension m. The way we do it is just by adding some coordinates. In a technical sense, solving lattice problems in a larger dimension is at least as hard as solving lattice problems in lower dimensions. No such reduction is known for the DLOG problem.

1.1.2. Lattices Background

An n dimensional lattice Λ is any subset of $\mathbb{R}^n$ that is both:

1. an *additive subgroup*: $\mathbf{0} \in \Lambda$ and $-\mathbf{x}, \mathbf{x}+\mathbf{y} \in \Lambda$ for every $\mathbf{x}, \mathbf{y} \in \Lambda$; and
2. *discrete*: every $\mathbf{x} \in \Lambda$ has a neighborhood in $\mathbb{R}^n$ in which $\mathbf{x}$ is the only lattice point.

Examples includes the integer lattice $\mathbb{Z}^n$, the scaled lattice $c\Lambda$ for any real number c and lattice Λ.

The *minimum distance* of a lattice Λ is the length of a shortest nonzero lattice vector:

$$\lambda_1(\Lambda) := \min_{\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}} \|\mathbf{v}\|.$$

(Unless otherwise specified $\|\cdot\|$ denotes the Euclidean norm.) More generally, the ith successive minimum $\lambda_i(\Lambda)$ is the smallest r such that Λ has i linearly independent vectors of norm at most r.

Because a lattice Λ is an additive subgroup of $\mathbb{R}^n$, we have the quotient group $\mathbb{R}^n/\Lambda$ of cosets

$$\mathbf{c} + \Lambda = \{\mathbf{c} + \mathbf{v} \ : \ \mathbf{v} \in \Lambda\}, \quad \mathbf{c} \in \mathbb{R}^n,$$

whith the usual induced addition operation operation operation $(\mathbf{c}_1+\Lambda)+(\mathbf{c}_2+\Lambda) = (\mathbf{c}_1+\mathbf{c}_2)+\Lambda$. A *fundamental domain* of Λ is a set $\mathcal{F} \subset \mathbb{R}^n$ that contains exactly one representative $\bar{\mathbf{c}} \in (\mathbf{c}+\Lambda) \cap \mathcal{F}$ of every coset $\mathbf{c}+\Lambda$. For example, the half-open intervals $[0,1)$ and $[-\frac{1}{2},\frac{1}{2})$ are fundamental domains of the integer lattice $\mathbb{Z}$, where coset $c+\mathbb{Z}$ has representative $c - \lfloor c \rfloor$ and $c - \lfloor c \rceil$ respectively.

Bases and fundamental parallelepipeds. Although every (non-trivial) lattice Λ is infinite, it is always finitely generated by the integer linear combinations of some linearly independent *basis* vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$:

$$\Lambda = \Lambda(\mathbf{B}) := \mathbf{B} \cdot \mathbb{Z}^k = \left\{ \sum_{i=1}^{k} z_i \mathbf{b}_i \ : \ z_i \in \mathbb{Z} \right\}.$$

The integer k is called the *rank* of the basis and is an invariant of the lattice. For the remainder of this manuscript we restrict our attention to *full-rank* lattices, where $k = n$. A lattice basis $\mathbf{B}$ is not unique: for any unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n \times n}$ (i.e. one having determinant ± 1), $\mathbf{B} \cdot \mathbf{U}$ is also a basis of $\Lambda(\mathbf{B})$, because $\mathbf{U} \cdot \mathbb{Z}^n = \mathbb{Z}^n$.

For a lattice Λ having basis $\mathbf{B}$, a commonly used fundamental domain is the origin-centered *fundamental parallelepiped* $\mathcal{P}(\mathbf{B}) := \mathbf{B} \cdot [-\frac{1}{2},\frac{1}{2})^n$ where coset $\mathbf{c}+\Lambda$ has representative $\mathbf{c} - \mathbf{B} \cdot \left\lfloor \mathbf{B}^{-1} \cdot \mathbf{c} \right\rceil$.

The dual lattice. The *dual* (sometimes called *reciprocal*) of a lattice $\Lambda \subset \mathbb{R}^n$ is defined as

$$\Lambda^* := \{\mathbf{w} \ : \ \langle \mathbf{w}, \Lambda \rangle \subseteq \mathbb{Z}\},$$

i.e. the set of points whose inner products with the vectors in Λ are all integers. It is straightforward to verify that Λ^* is a lattice. For example $(\mathbb{Z}^n)^* = \mathbb{Z}^n$ and $(c\Lambda)^* = c^{-1}\Lambda^*$ for any nonzero real c and lattice Λ. It is also easy to verify that if $\mathbf{B}$ is a basis of Λ, then $\mathbf{B}^{-t} := (\mathbf{B}^t)^{-1} = (\mathbf{B}^{-1})^t$ is a basis of Λ^*.

Computational problems. We now define some of the intensively studied computational problems on lattices which appear to be difficult (except for very large approximation factors) and that have been most useful in cryptography. Perhaps the most well-studied computational problem on lattices is the *shortest vector problem*:

Definition 1.1 (Shortest Vector Problem (SVP))**.** *Given an arbitrary basis* $\mathbf{B}$ *of some lattice* $\Lambda = \Lambda(\mathbf{B})$*, find a shortest nonzero lattice vector, i.e. a* $\mathbf{v} \in \Lambda$ *for which* $\|\mathbf{v}\| = \lambda_1(\Lambda)$*.*

Particularly important to lattice cryptography are *approximation* problems, which are parameterized by an approximation factor $\gamma \geq 1$ that is typically taken to be a function of the lattice dimension n, i.e. $\gamma = \gamma(n)$. For example, the approximation version of SVP is as follows (note that by setting $\gamma(n) = 1$ we recover the problem defined above):

Definition 1.2 (Approximate Shortest Vector Problem (SVP_γ))**.** *Given a basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$*, find a nonzero lattice vector* $\mathbf{v} \in \Lambda$ *for which* $\|\mathbf{v}\| \leq \gamma(n) \cdot \lambda_1(\Lambda)$*.*

As described below, several cryptosystems can be proved secure assuming the hardness of certain lattice problems, in the worst case. However, to date no such proof is known for the *search* version of SVP_γ. Instead, there are proofs based on the following *decision* version of approximate-SVP, as well as a *search* problem related to the nth successive minimum:

Definition 1.3 (Decisional Approximate SVP (GapSVP$_\gamma$))**.** *Given a basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$ *where either* $\lambda_1(\Lambda) \leq 1$ *or* $\lambda_1(\Lambda) > \gamma(n)$*, determine which is the case.*

Definition 1.4 (Approximate Shortest Independent Vectors Problem (SIVP$_\gamma$))**.** *Given a basis* $\mathbf{B}$ *of a full-rank* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$*, output a set* $\mathbf{S} = \{\mathbf{s}_i\} \subset \Lambda$ *of* n *linearly independent lattice vectors where* $\|\mathbf{s}_i\| \leq \gamma(n) \cdot \lambda_n(\Lambda)$ *for all* i*.*

A final important problem for cryptography is the following *bounded-distance decoding* (BDD) problem, which asks to find the lattice vector that is closest to a given target point $\mathbf{t} \in \mathbb{R}^n$, where the target is promised to be "rather close" to the lattice. This promise, and the uniqueness of the solution, are what distinguish BDD$_\gamma$ from the approximate *closest vector problem* CVP$_\gamma$, wherein the target can be an arbitrary point.

Definition 1.5 (Bounded Distance Decoding Problem (BDD$_\gamma$))**.** *Given a basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$ *and a target point* $\mathbf{t} \in \mathbb{R}^n$ *with the guarantee that* $\mathsf{D}(\mathbf{t}, \Lambda) < d = \lambda_1(\Lambda)/(2\gamma(n))$*, find the unique lattice vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{t} - \mathbf{v}\| < d$*.*

1.1.3. Early Constructions

NTRU encryption. In a concurrent work with Ajtai's in 1996, Hoffstein, Pipher and Silverman [HPS98] devised the public-key encryption scheme NTRUEncrypt. This was the first practical cryptosystem in lattice-based cryptography thanks to the use of polynomial rings, which is interpreted in terms of algebraically structured lattices. Moreover, NTRUEncrypt has quite compact keys and it has withstood significant cryptanalysis efforts when appropriately parametrized. Unlike Ajtai's constructions, there is no known reduction from any worst-case lattice problem to any standard version of the NTRU problem. However, a variant of NTRUEncrypt has been proved secure [SS11], assuming the hardness of ring-LWE (see Definition 1.10).

NTRUEncrypt is parameterized by a polynomial ring $R = \mathbb{Z}[X]/(f(X))$, with $f(X) = X^n - 1$ for a pime n or $f(X) = X^n + 1$ for an n that is a power of two, and a sufficiently large odd modulus q that defines the quotient ring $R_q = R/qR$. The public key is $h = 2g \cdot s^{-1} \in R_q$ for two "short" polynomials $g, s \in R$, i.e. ones having relatively small integer coefficients, where the secret key s is also chosen to be invertible modulo both q and two. Encryption essentially involves multiplying h by a short "blinding" factor $r \in R$ and adding a short error term $e \in R$ that encodes the message bits in its coefficients modulo two, to get a cithertext $c = h \cdot r + e \in R_q$. Decryption is done by multiplying the ciphertext by the secret key to get $c \cdot s = 2g \cdot r + e \cdot s \in R_q$ and interpreting the result as a short element of R, which works because all of g, r, e and s are short. From this, one recovers $e \cdot s$ modulo two, and thereby e modulo two, to recover the message bits. Note that there are more efficient variants of this, e.g. choosing $s = 1 \pmod 2$, so that $e \cdot s = e \pmod 2$.

Goldreich-Goldwasser-Halevi encryption and signatures. Inspired by Ajtai's seminal work [Ajt96] along with McEliece's code-based cryptosystem [McE78], Goldreich, Goldwasser, Halevi (GGH) [GGH97] proposed a public-key encryption scheme and digital signature scheme based on lattice problems. Unlike the works of Ajtai, the GGH proposals did not come with any worst-case security guarantees; their conjectured security was merely heuristic. This encryption scheme was successfully cryptanalyzed for practical parameter sizes (but not broken asymptotically) [Ngu99], and the GGH signature scheme was later broken completely [NR06]. However, the central ideas underlying the GGH proposals were later resurrected and instantiated in ways that admit security proofs under worst-case hardness assumptions, and have subsequently led to an enormous variety of applications.

The main idea behind GGH encryption and signatures is that a public key is a "bad" basis of some lattice, while the corresponding secret key is a "good" basis of the same lattice. Roughly speaking, a "bad" basis is one consisting of long and highly non-orthogonal lattice vectors, while a "good" basis consists of relatively short lattice vectors. Notice that with a good basis one can solve BDD_γ, where γ is related to the norm of the good basis vectors, while, given a bad basis, solving BDD_γ is a hard problem. Such bases can be generated together, e.g. by first choosing the good basis and then multiplying it by some randomly chosen unimodular transformation (which preserves the lattice) to obtain the bad basis. Alternatively, every integer lattice has a special basis, called the *Hermite normal form*, which is in a precise sense a "hardest possible" basis for the lattice, because it can be efficiently computed from any other basis. So the Hermite normal form is a best-possible choice for the public basis [Mic01].

In the GGH encryption scheme, the sender uses the public key to choose a "random" lattice point $\mathbf{v} \in \Lambda$ that somehow encodes the message, and then adds to it some small error $\mathbf{e} \in \mathbb{R}^n$, letting the ciphertext be $\mathbf{c} = \mathbf{v} + \mathbf{e} \in \mathbb{R}^n$. The error is small enough to ensure that $\mathbf{c}$ is much closer to $\mathbf{v}$ than to any other lattice point, so the ciphertext unambiguously represents the message, and recovering $\mathbf{v}$ from $\mathbf{c}$ is a random instance of the *bounded-distance decoding* (BDD) problem. The receiver, using its knowledge of the good basis, can easily decode $\mathbf{c}$ back to $\mathbf{v}$ and recover the message. For security, one may conjecture that an eavesdropper who knows only the bad basis cannot decode $\mathbf{c}$, or even learn anything about $\mathbf{v}$, which implies that the message is hidden.

In the GGH signature scheme, a message to be signed is mapped to a point $\mathbf{m} \in \mathbb{R}^n$, e.g. by a suitable public hash function. The signer then uses its good basis to find a lattice vector $\mathbf{v} \in \Lambda$ relatively close to $\mathbf{m}$, which serves as the signature. A verifier, using only the public bad basis, can verify that $\mathbf{v}$ is a lattice vector and is sufficiently close to $\mathbf{m}$. For security, one may conjecture that a forger who knows only the bad basis and some previous message-signature pairs cannot find a lattice vector sufficiently close to $\mathbf{m}'$ for an unsigned message $\mathbf{m}'$. It turns out, however, that this conjecture is *false*, as shown most severely in [NR06]. The main problem is that signatures leak significant information about the geometry of the secret good basis, and after a relatively small number of signatures, an adversary can eventually recover the secret basis entirely, allowing it to forge signatures for arbitrary messages.

NTRU meets GGH. Following the ideas in [GGH97], compact ring-based instantiations using NTRU-type lattices, known as NTRUSign, were proposed in [HHP+03]. These were subject to various practical attacks, in addition to the generic ones that apply to all GGH-type signatures. The second proposal [HHP+03] includes a "perturbation" technique that is intended to make signatures reveal significantly less information about the secret key at the cost of larger keys and parameters. The main idea is that the algorithm that decodes the (hashed) message $\mathbf{m} \in \mathbb{R}^n$ to a nearby lattice vector $\mathbf{v} \in \Lambda$ is substantially less linear, because it involves two unrelated lattice bases. However, the ideas of [NR06] were extended to also break this variant [DN12b].

1.2. Modern Lattice-based Cryptography

Following the seminal work of Ajtai [Ajt96], worst-case to average-case reductions for lattice problems were proved forming, with some Gaussian-like probability distributions over lattices, the foundations of the modern lattice-based cryptography.

A particularly interesting modern construction in lattice-based cryptography is a framework using these Gaussian-like probability distribution over lattices to construct provably secure hash-

and-sign signature schemes. Indeed, inspired by the early ideas of Goldreich, Goldwasser and Halevi (GGH) [GGH97], Gentry, Peikert and Halevi (GPV) [GPV08] showed that certain types of trapdoor functions can be constructed from lattice problems, and in particular (ring-)SIS/LWE (see the definitions 1.6, 1.7 and 1.10), thereby correcting the leakage problem of the GGH signature scheme.

The powerful notion of *fully homomorphic encryption* (FHE), first envisioned by Rivest, Adleman and Dertouzos [RAD78], allows an untrusted worker to perform arbitrary computations on encrypted data, without learning anything about that data. For three decades FHE remained an elusive "holy grail" goal, until Gentry [Gen09] proposed the first candidate construction of FHE, which was based on lattices (as were all subsequent constructions), and more precisely on problems using again Gaussian-like distributions. More recently, lattices have provided the only known realizations of other versatile and powerful cryptographic notions, such as attribute-based encryption for arbitrary access policies [GVW13; BGG+14] and general-purpose code obfuscation [GGH+13].

1.2.1. Modern Foundations

The *short integer solution* (SIS) problem was first introduced in the seminal work of Ajtai [Ajt96], and has served as the foundation for one-way and collision-resistant hash functions, identification schemes, digital signatures, and other so-called "minicrypt" primitives (but not public-key encryption). Informally, the SIS problem asks, given many uniformly random elements of a certain large finite additive group, to find a sufficiently "short" nontrivial integer combination of them that sums to zero. More formally, SIS is parameterized by positive integers n and q defining the group $\mathbb{Z}_q^n$, a positive real β, and a number m of group elements. For concreteness, one should think of n as being the main hardness parameter (e.g. $n \geq 100$), and $q > \beta$ being a (small) polynomial in n. The parameter m is of secondary importance, so we sometimes leave it unspecified.

Definition 1.6 (Short Integer Solution ($\mathrm{SIS}_{n,q,\beta,m}$))**.** *Given m uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$, forming the columns of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, find a nonzero integer vector $\mathbf{z} \in \mathbb{Z}^m$ of norm $\|\mathbf{z}\| \leq \beta$ such that*

$$f_{\mathbf{A}}(\mathbf{z}) := \mathbf{A}\mathbf{z} = \sum_i \mathbf{a}_i \cdot z_i = \mathbf{0} \in \mathbb{Z}_q^n.$$

The SIS problem can be seen as an *average-case* short-vector problem on a certain family of so-called "q-ary" m-dimensional integer lattices, namely, the lattices

$$\Lambda^{\perp}(\mathbf{A}) := \left\{\mathbf{z} \in \mathbb{Z}^m \ : \ \mathbf{A}\mathbf{z} = \mathbf{0} \in \mathbb{Z}_q^n\right\} \supseteq q\mathbb{Z}^m.$$

Borrowing the terminology of coding theory, here $\mathbf{A}$ acts as a "parity-check" matrix that defines the lattice $\Lambda^{\perp}(\mathbf{A})$, where $\mathbf{A}$ is chosen uniformly at random.

One can also consider an *inhomogeneous* version of the SIS problem, which is to find a short integer solution to $\mathbf{A}\mathbf{x} = \mathbf{u} \in \mathbb{Z}_q^n$, where $\mathbf{A}$, $\mathbf{u}$ are uniformly random and independent. It is not hard to show that the homogeneous and inhomogeneous problems are essentially equivalent for typical parameters.

Starting from Ajtai's seminal work [Ajt96], a long sequence of works has established progressively stronger results about the hardness of the SIS problem relative to worst-case lattice problems. All such results are instances of the following template:

Theorem 1.1. *For any $m = \mathrm{poly}(n)$, any $\beta > 0$ and any sufficiently large $q \geq \beta \cdot \mathrm{poly}(n)$, solving $\mathrm{SIS}_{n,q,\beta,m}$ with non-negligible probability is at least as hard as solving the decisional approximate*

shortest vector problem GapSVP$_\gamma$ and the approximate shortest independent vectors problems SIVP$_\gamma$ (among others) on arbitrary n-dimensional lattices (i.e. in the worst case) with overwhelming probability, for some $\gamma = \beta \cdot \textit{poly}(n)$.

Notice that the exact values of m and q (apart from its lower bound) play essentially no role in the ultimate hardness guarantee, but that the approximation factor γ degrades with the norm bound β on the SIS solution. Inspired by the ideas behind NTRUEncrypt [HPS98], Micciancio [Mic02] introduced a compact ring-based analogue of Ajtai's SIS problem. This analogue has come to be known as the *ring*-SIS problem.

A very important work of Regev [Reg05] from 2005 introduced the average-case *learning with errors* (LWE) problem, which is the "encryption-enabling" analogue of the SIS problem. Indeed, the two problems are syntactically very similar, and can meaningfully be seen as duals of each other. LWE is parameterized by positive integers n and q, and an error distribution χ over $\mathbb{Z}$. For concreteness, n and q can be thought of as roughly the same as is SIS, and χ is usually taken to be a discrete Gaussian of width αq for some $\alpha < 1$, which is often called the relative "error rate".

Definition 1.7 (LWE distribution)**.** *For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ called the* secret*, the LWE distribution $A_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e \mod q)$.*

There are two main versions of the LWE problem: *search*, which is to find the secret given LWE samples, and *decision*, which is to distinguish between LWE samples and uniformly random ones. We additionally parameterize these problems by the number m of available samples, which we typically take to be large enough that the secret is uniquely defined with high probability. As with SIS, the parameter m is of secondary importance, so we often leave it unspecified.

Definition 1.8 (Search-LWE$_{n,q,\chi,m}$)**.** *Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ drawn from $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), find $\mathbf{s}$.*

Definition 1.9 (Decision-LWE$_{n,q,\chi,m}$)**.** *Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where every sample is distributed according to either: (1) $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), or (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).*

Regev proved the following worst-case hardness theorem for LWE (stated here in a slightly stronger form from [Pei16]):

Theorem 1.2 ([Reg05])**.** *For any $m = \textit{poly}(n)$, any modulus $q \leq 2^{\textit{poly}(n)}$ and any (discretized) Gaussian error distribution χ of parameter $\alpha q \geq 2\sqrt{n}$ where $0 < \alpha < 1$, solving the decision-LWE$_{n,q,\chi,m}$ problem is at least as hard as quantumly solving GapSVP$_\gamma$ and SIVP$_\gamma$ on arbitrary n-dimensional lattices, for some $\gamma = \tilde{\mathcal{O}}(n/\alpha)$.*

Notice that, just as in the worst-case hardness theorem for SIS, the exact values of m and q (apart from its lower bound of $2\sqrt{n}/\alpha$) play essentially no role in the ultimate hardness guarantee. However, the approximation factor γ degrades with the inverse error rate $1/\alpha$ of the LWE problem.

In work published in 2010, Lyubashevsky, Peikert and Regev [LPR10] introduced *ring*-LWE, the ring-based analogue of LWE and proved the hardness theorems described below. Ring-LWE is parameterized by a ring R of degree n over $\mathbb{Z}$, a positive integer modulus q defining the quotient ring $R_q = R/qR$ and an error distribution χ over R. Typically, one takes R to be a *cyclotomic* ring, i.e. $\mathbb{Z}[X]/\Phi_d(X)$ with $\phi(d) = n$ and where $\Phi_d(X)$ is the nth cyclotomic polynomial, and χ to be some kind of discretized Gaussian in the canonical embedding of R, which we can roughly think of as having an "error rate" $\alpha < 1$ relative to q.

Definition 1.10 (Ring-LWE distribution). *For an $s \in R_q$ called the* secret*, the ring-LWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, choosing $e \leftarrow \chi$ and outputting $(a, b = s \cdot a + e \mod q)$.*

The decision version of the R-LWE problem is to distinguish between ring-LWE samples and uniformly random ones. As usual, we also parameterize the problem by the number m of available samples, which is sometimes left unspecified.

Just as in LWE, without errors the ring-LWE problem is easy, because in case (1) we can efficiently find s: given a sample (a_i, b_i) where $a_i \in R_q$ is invertible (most elements of R_q are), we have $s = b_i \cdot a_i^{-1}$, whereas in case (2) there will almost never be a single s that is consistent with all samples. (Ring-)LWE has a *normal form*, in which the secret s is chosen from the error distribution (modulo q) rather than uniformly.

The primary advantage of ring-LWE is its compactness and efficiency: each sample (a_i, b_i) yields an n-dimensional pseudorandom ring element $b_i \in R_q$, rather than just a single pseudorandom scalar $b_i \in \mathbb{Z}_q$ as in LWE. In addition, ring multiplication can be performed in only quasi-linear $\tilde{\mathcal{O}}(n)$ time using FFT-like techniques, so we can generate these n pseudorandom scalar in just $\tilde{\mathcal{O}}(1)$ amortized time each. For example, this all yields a public-key encryption scheme with only $\tilde{\mathcal{O}}(1)$-factor overheads in encryption/decryption time and ciphertext space, versus sending the plaintext in the clear.

Like LWE, ring-LWE enjoys a worst-case hardness guarantee, informally stated here:

Theorem 1.3 ([LPR10]). *For any $m = \mathsf{poly}(n)$, cyclotomic ring R of degree n (over $\mathbb{Z}$), and appropriate choices of modulus q and error distribution χ of error rate $\alpha < 1$, solving the R-LWE$_{q,\chi,m}$ problem is at least as hard as quantumly solving the SVP$_\gamma$ problem on arbitrary ideal lattices in R, for some $\gamma = \mathsf{poly}(n)/\alpha$.*

Notice that as with LWE, the approximation factor γ varies inversely with the error rate α of χ. Unlike with LWE, however, the factor γ also degrades slightly with the number of samples m. This degradation may be an artifact of the proof technique, and in any case it can be avoided by choosing the error distribution *itself* at random from a certain family.

Recall from Section 1.1.2 that the NTRU cryptosystem of Hoffstein, Pipher and Silverman [HPS98] was an early lattice-based cryptographic proposal. Several computational problems naturally relate to the NTRU system. One such problem is the following:

Definition 1.11 (NTRU learning problem). *For an invertible $s \in R_q^*$ and a distribution χ on R, define $N_{s,\chi}$ to be the distribution that outputs $e/s \in R_q$ where $e \leftarrow \chi$. The* NTRU learning problem *is: given independent samples $a_i \in R_q$ where every sample is distributed according to either: (1) $N_{s,\chi}$ for some randomly chosen $s \in R_q^*$ (fixed for all samples), of (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).*

The NTRU and ring-LWE problems are syntactically very similar and can even be viewed as homogeneous and inhomogeneous versions of the same problem. Specifically, for NTRU samples $a_i \in R_q$ there is a secret s such that every $a_i \cdot s = e_i \pmod q$ for some short $e_i \in R$, while for ring-LWE samples $(a_i, b_i) \in R_q \times R_q$, there is a secret s such that every $a_i \cdot s + b_i = e_i \pmod q$ for some short $e_i \in R$.

The problems presented in this section (an many modern works on lattices in complexity and cryptography) rely on Gaussian-like probability distributions over lattices, called *discrete Gaussians*. Here we recall the relevant definitions.

Gaussians. For any positive integer n and real $s > 0$, which is taken to be $s = 1$ when omitted, define the *Gaussian function* $\rho_s : \mathbb{R}^n \to \mathbb{R}^+$ *of parameter (or width)* s *as*

$$\rho_s(\mathbf{x}) := \exp(-\pi\|\mathbf{x}\|^2/s^2) = \rho(\mathbf{x}/s).$$

Notice that ρ_s is invariant under rotations of $\mathbb{R}^n$ and that $\rho_s(\mathbf{x}) = \prod_{i=1}^n \rho_s(x_i)$ and that for $x \in \mathbb{Z}$ and $c \in \mathbb{R}$ we extend this definition to $\rho_{s,c}(x) := \rho_s(x - c)$.

The (continuous) Gaussian distribution $D_{\mathbb{R}^n,s}$ of parameter s over $\mathbb{R}^n$ is defined to have probability density function proportional to ρ_s, i.e.

$$D_{\mathbb{R}^n,s}(\mathbf{x}) := \rho_s(\mathbf{x}) / \int_{\mathbb{R}^n} \rho_s(\mathbf{z})\, d\mathbf{z} = \rho_s(\mathbf{x})/s^n.$$

For a lattice coset $\mathbf{c} + \Lambda \subset \mathbb{R}^n$ with $\mathbf{c} \in \mathbb{R}^n$ and parameter $s > 0$, the *discrete Gaussian probability* distribution $D_{\mathbf{c}+\Lambda,s}$ is simply the Gaussian distribution restricted to the coset:

$$D_{\mathbf{c}+\Lambda,s}(\mathbf{x}) \propto \begin{cases} \rho_s(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{c} + \Lambda \\ 0 & \text{otherwise.} \end{cases}$$

For one-dimensional discrete Gaussian distributions we omit the first parameter and add the center $c \in \mathbb{R}$ as parameter, i.e. for any $x \in \mathbb{Z}$:

$$D_{s,c}(x) := D_{c+\mathbb{Z},s}(x) = \rho_{s,c}(x) / \sum_{y \in \mathbb{Z}} \rho_{s,c}(y).$$

Smoothing parameter. Micciancio and Regev [MR04] introduced a very important quantity called the *smoothing parameter* of a lattice Λ. Informally, this is the amount of Gaussian "blur" required to "smooth out" essentially all the discrete structure of Λ. In other words, the smoothing parameter $\eta_\epsilon(\Lambda)$ quantifies the minimal discrete Gaussian parameter $s > 0$ required to obtain a given level of smoothness on the lattice Λ, if one picks a noise vector over a lattice from a discrete Gaussian distribution with width parameter at least as large as the smoothing parameter, and reduces this modulo the fundamental parallelepiped of the lattice, then the resulting distribution is very close to uniform on the fundamental parallelepiped. Alternatively, it can be seen as the smallest width parameter $s > 0$ such that every coset $\mathbf{c} + \Lambda$ has nearly the same Gaussian mass $\rho_s(\mathbf{c} + \Lambda) := \sum_{\mathbf{x} \in \mathbf{c}+\Lambda} \rho_s(\mathbf{x})$, up to some small relative error.

Formally, the smoothing parameter $\eta_\epsilon(\Lambda)$ is parameterized by a tolerance $\epsilon > 0$ and is defined using the dual lattice as the minimal $s > 0$ such that $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \varepsilon$. This condition can be used to formalize and prove the above-described "smoothing" properties. Notice that from [MR07, Lemma 3.3] for any n-dimensional lattice Λ and positive real $\varepsilon > 0$

$$\eta_\epsilon(\Lambda) \leq \sqrt{\frac{\log(2n(1 + 1/\epsilon))}{\pi}} \cdot \lambda_n(\Lambda).$$

1.2.2. Provably Secure Lattice Signatures from Gaussian Sampling

Informally, a *trapdoor function* is a function that is easy to evaluate and hard to invert on its own, but which can be generated together with some extra "trapdoor" information that makes inversion easy. There are many versions of this basic concept, depending on whether the function in question is injective, surjective, bijective, "lossy", etc. The prototypical candidate trapdoor function is the

RSA function [RSA78] $f_{N,e}(x) = x^e \mod N$, where N is the product of distinct primes p,q and $\gcd(e, \phi(N)) = 1$. The RSA function is a bijection on $\mathbb{Z}_N^*$, and the trapdoor is $d = e^{-1} \mod \phi(N)$, because $(x^e)^d = x \mod N$. Alternatively, the factorization p, q of N is a trapdoor, because one can efficiently compute d from these factors. There are relatively few trapdoor candidates, all commonly accepted ones relied on the conjectured hardness of integer factorization [RSA78; Rab79; Pai99].

Inspired by the early ideas of Goldreich, Goldwasser and Halevi (GGH) [GGH97], Gentry, Peikert and Halevi (GPV) [GPV08] showed that certain types of trapdoor functions can be constructed from lattice problems, and in particular (ring-)SIS/LWE. The work of GPV and several follow-ups used these trapdoor functions to construct many powerful cryptographic applications, including digital signature schemes, (hierarchical) identity-based and attribute-based encryption, and much more.

In the GGH signature scheme, signing a message roughly corresponds to solving an approximate *closest vector problem* (CVP) on the average, which can be accomplished using a short basis. In this case, the target can be an arbitrary point that not be especially close to the lattice, and it has many valid signatures corresponding to sufficiently nearby lattice vectors. Recall, however, that the GGH signature scheme and some of its derivatives (e.g. NTRUSign) turned out to be insecure [NR06], because an attacker use a small number of signatures to efficiently reconstruct the secret trapdoor basis. This is because the signing algorithm implicitly leaks information about the geometry of the secret basis it uses.

The work of GPV gave a different, randomized approach to signing that provably leaks *no information* about the secret basis (apart from a bound on its length, which is already public knowledge). A key property is that the probability distribution of a signature is the same *no matter which short basis is used to produce it.* This fact facilitates a formal proof of unforgeability in the random-oracle model, assuming the average-case hardness of a CVP-like problem as SIS.

In parallel with the above trapdoor technique, applying the Fiat-Shamir heuristic [FS87] was also explored for lattice-based signatures [LM08; Lyu08; Lyu09; Lyu12; DDLL13]. Notice that both paradigms achieve comparable levels of compactness, but hash-and-sign has interesting properties: GPV comes with a security proof in the random oracle [GPV08], and a security proof in the quantum random oracle model was later provided in [BDF+11]. This stands in contrast with schemes using the Fiat-Shamir heuristic, which are notoriously harder to render secure in the QROM [KLS17; Unr17]. In addition, it enjoys message-recovery capabilities [PLP16], its verification procedure is very simple, and it can be converted into an IBE scheme in a straightforward manner.

1.2.2.1. Gaussian Sampling over a Lattice Coset

A key technical ingredient behind the GPV approach is an algorithm for sampling from a discrete Gaussian distribution over a lattice coset, given any sufficiently good basis of the underlying lattice. More precisely, letting $\widetilde{\mathbf{S}} = \{\widetilde{\mathbf{s}}_i\}$ denote the Gram-Schmidt orthogonalization of an ordered set of vectors $\mathbf{S} = \{\mathbf{s}_i\}$ and letting $\|\mathbf{B}\| := \max_i \|\mathbf{b}_i\|$ for any set of vectors $\mathbf{B} = \{\mathbf{b}_i\}$, we have:

Theorem 1.4 ([GPV08]). *There is a randomized polynomial-time algorithm that, given any basis* $\mathbf{S}$ *of a lattice* $\Lambda = \Lambda(\mathbf{S}) \subset \mathbb{R}^n$*, any coset* $\mathbf{c} + \Lambda$ *and any Gaussian parameter* $s \geq \|\widetilde{\mathbf{S}}\| \cdot \sqrt{\log \mathcal{O}(n/\varepsilon)}$*, outputs a sample whose distribution is within statistical distance* ε *of the discrete Gaussian* $D_{\mathbf{c}+\Lambda,s}$*.*

The sampling algorithm from Theorem 1.4 is a randomized variant of Babai's "nearest-plane" algorithm [Bab86], where in each iteration we randomly choose a "plane" according to a one-dimensional discrete Gaussian over an appropriate coset $c + \mathbb{Z}$, instead of deterministically. Interestingly, Klein [Kle00] proposed first the same randomized variant of nearest-plane, but for the

problem of *bounded-distance decoding* (BDD). Subsequent works gave other sampling algorithms that offer various trade-offs among efficiency, statistical error and width of the sampled distribution.

Since the introduction of discrete Gaussian sampling for cryptographic purposes [GPV08], many refinements and alternative algorithms have been proposed: we discuss some of them here, but these sampling algorithms are discused in more detail in Section 3.1.2. The randomized nearest-plane algorithm for discrete Gaussian sampling described in [GPV08] is rather inefficient in general: it requires very high-precision arithmetic to store and operate on an appropriate representation of the Gram-Schmidt orthogonalized basis and it involves n sequential iterations, each of which requires an n-dimensional inner product of high-precision vectors. Also, this super-quadratic runtime persists to the ring setting, where by contrast all other commonly used ring operations have a only quasi-linear cost.

Peikert [Pei10] gave a more efficient and parallel discrete Gaussian sampling algorithm, which has a quasi-linear cost in the ring setting. In the same way as the algorithm described in [GPV08] this new algorithm is a randomized variant of Babai's "round-off" algorithm [Bab86]. As an illustrative first attempt, to sample in parallel from $\mathbf{c} + \Lambda$ with $\mathbf{c} \in \mathbb{R}^n$ using a short basis $\mathbf{S}$ of Λ, one might try randomly rounding each coefficient of the basis vectors *independently*, i.e. let $\mathbf{x} \leftarrow \mathbf{S} \cdot D_{S^{-1}\mathbf{c}+\mathbb{Z}^n,r}$ for some appropriate $r \geq \eta_\varepsilon(\mathbb{Z}^n)$. This produces a vector close to $\mathbf{c}$, but unfortunately, the distribution is "skewed" due to the multiplication by $\mathbf{S}$. More formally, it is a *non-spherical* discrete Gaussian with covariance matrix $\mathbb{E}[\mathbf{x} \cdot \mathbf{x}^t] \approx r^2 \cdot \mathbf{S}\mathbf{S}^t$, which reveals a lot about $\mathbf{S}$. The main idea in [Pei10] is to add an appropriately distributed *perturbation* term to "unskew" the distribution of $\mathbf{x}$, making it spherical. Its primary disadvantage is that the vectors it produce are longer in average, which corresponds to worse security (and thus larger keys and signatures to compensate).

1.2.3. Fully Homomorphic Encryption

In 1978, Rivest, Adleman and Dertouzos [RAD78] proposed a concept which has come to be known as *fully homomorphic encryption* (FHE). In brief, an FHE scheme allows *computation on encrypted data*, or more concisely, *homomorphic computation*: given a ciphertext that encrypts some data μ, one can compute a ciphertext that encrypts $f(\mu)$ for any desired (efficiently computable) function f. We emphasize that this is possible without ever needing to decrypt the data or know the decryption key.

Fully homomorphic encryption was known to have abundant applications in cryptography, but for three decades no plausibly secure scheme was known. This changed in 2009, when Gentry proposed a candidate FHE scheme based on ideal lattices [Gen09]. Gentry's seminal work generated tremendous excitement, and was quickly followed by many works (e.g. [DGHV10; Gen10; SV14; BV11b; CMNT11; BV11a; BGV12; CNT12; GHS12b; Bra12; GHPS12; CCK+13; GSW13]), that offered various improvements in conceptual and technical simplicity, efficiency, security guarantees, etc.

FHE from LWE. The earliest "first generation" FHE constructions [Gen09; DGHV10] were based on ad-hoc average-case assumptions about ideal lattices and the "approximate GCD" probelm, respectively. In a sequence of works, Brakerski and Vaikuntanathan (BV) [BV11b; BV11a] gave a "second generation" of FHE constructions, which were based on standard assumptions supported by worst-case hardnes, namely, (ring-)LWE. Here we describe the main idea behind the LWE-based scheme from [BV11a], with additional improvements from a subsequent work with Gentry [BGV12].

The BV scheme encrypts a single bit per ciphertext, and supports homomorphic *addition* and *multiplication* modulo two. Notice that this scheme is easily generalizable to a message space of $\mathbb{Z}_p$ for any integer p. This is sufficient for FHE because by composing such operations, one can

homomorphically evaluate any boolean circuit. In the BV system, a secret key is an LWE secret, and an encryption of a bit is simply an LWE sample for an odd modulus q, where the error term e encodes the message as its least-significant bit. More precisely, a cipher that encrypts $\mu \in \mathbb{Z}_2$ under a secret key $\mathbf{s} \in \mathbb{Z}^n$ is a vector $\mathbf{c} \in \mathbb{Z}_q^n$ such that

$$\langle \mathbf{s}, \mathbf{c} \rangle = \mathbf{s}^t \cdot \mathbf{c} = e \pmod{q}, \tag{1.1}$$

where $e \in \mathbb{Z}$ is some small error such that $e = \mu \pmod 2$, i.e. $e \in \mu + 2\mathbb{Z}$. To decrypt $\mathbf{c}$ using $\mathbf{s}$, one just computes $\langle \mathbf{s}, \mathbf{c} \rangle \in \mathbb{Z}_q$, lifts the result to its unique representative $e \in \mathbb{Z} \cap [-\frac{q}{2}, \frac{q}{2})$ and outputs $\mu = e \mod 2$.

Notice that by taking $\mathbf{s} = (-\bar{\mathbf{s}}, 1)$, a ciphertext vector $\mathbf{c} = (\bar{\mathbf{c}}, c)$ is just an LWE sample with an $(n-1)$-dimensional secret $\bar{\mathbf{s}}$ and error term e, because $c = \langle \bar{\mathbf{s}}, \bar{\mathbf{c}} \rangle + e \pmod q$. In the symmetric-key setting, such a ciphertext can be produced directly using $\bar{\mathbf{s}}$; in the asymmetric-key setting we can just add a random combination of LWE samples with respect to $\bar{\mathbf{s}}$, which are given in the public key. Note that this is essentially how all LWE public-key encryption schemes work. Using these observations it is straightforward to show that ciphertexts are pseudorandom, hence the encryption is passively secure, assuming the hardness of decision-LWE.

We remark that the decryption relation expressed in Equation 1.1, where $e \in \mu + 2\mathbb{Z}$, is sometimes called the "least significant bit" encoding of the message, as opposed to the "most significant bit" encoding where $\langle \mathbf{s}, \mathbf{c} \rangle \approx \mu \cdot \lfloor \frac{q}{2} \rceil \mod q$). It turns out that the two losslessly switch between them without knowing the secret key.

Homomorphic operations. We now describe homomorphic addition and multiplication. For $i = 1, 2$, let $\mathbf{c}_i \in \mathbb{Z}_q^n$ be a ciphertext that encrypts $\mu_i \in \mathbb{Z}_2$ under secret key $\mathbf{s}$, with small error term $e_i \in \mu_i + 2\mathbb{Z}$. Homomorphic addition is simple: $\mathbf{c}_1 + \mathbf{c}_2 \in \mathbb{Z}_q^n$ encrypts $\mu_1 + \mu_2 \in \mathbb{Z}_2$, because

$$\langle \mathbf{s}, \mathbf{c}_1 + \mathbf{c}_2 \rangle = \langle \mathbf{s}, \mathbf{c}_1 \rangle + \langle \mathbf{s}, \mathbf{c}_2 \rangle = e_1 + e_2 \pmod q,$$

and of course $e_1 + e_2 \in \mu_1 + \mu_2 + 2\mathbb{Z}$ is still small. Notice, however, that we cannot add an unbounded number of ciphertexts, because eventually the magnitude of the error will grow larger than $q/2$, in which case decryption may fail; we return to this issue shortly.

Homomorphic multiplication is a bit trickier, the idea is to use the *tensor* (or Kronecker) product. Note that this idea has been firstly studied by Aguilar-Melchor, Gaborit and Herranz in the late published article [MGH10]. We start with the observation that the tensor product $\mathbf{c}_1 \otimes \mathbf{c}_2 = (c_{1,i} \cdot c_{2,j})_{i,j} \in \mathbb{Z}_q^{n^2}$ is a valid encryption of $\mu_1\mu_2 \in \mathbb{Z}_2$ under an *alternative secret key* $\mathbf{s} \otimes \mathbf{s} \in \mathbb{Z}_q^{n^2}$, i.e. the secret key tensored with itself. This is because by the mixed-product property of tensor products

$$\langle \mathbf{s} \otimes \mathbf{s}, \mathbf{c}_1 \otimes \mathbf{c}_2 \rangle = \langle \mathbf{s}, \mathbf{c}_1 \rangle \cdot \langle \mathbf{s}, \mathbf{c}_2 \rangle = e_1 \cdot e_2 \pmod q,$$

and $e_1 \cdot e_2 \in \mu_1\mu_2 + 2\mathbb{Z}$ is still rather small, as long as the original errors were small enough to begin with. So just as with homomorphic addition, the number of homomorphic multiplications is bounded a priori.

Key switching. Homomorphic multiplication as described above has an even more significant problem than the error growth: the *dimension* of the cithertext also grows extremely fast, i.e. exponentially with the number of multiplied cithertexts, due to the use of the tensor product. To resolve the issue, BV introduced a clever *dimension reduction* — also called *key switching* — technique.

Suppose we have an n_{in}-dimensional ciphertext $\mathbf{c}_{\text{in}}$ (e.g. $_{\text{in}} = \mathbf{c}_1 \otimes \mathbf{c}_2$ as above) that encrypts some message μ under a secret key $\mathbf{s}_{\text{in}}$ (e.g. $\mathbf{s}_{\text{in}} = \mathbf{s} \otimes \mathbf{s}$ as above), under the "most-significant bit" encoding:

$$\langle \mathbf{s}_{\text{in}} \otimes \mathbf{c}_{\text{in}} \rangle = \mathbf{s}_{\text{in}}^t \cdot \mathbf{c}_{\text{in}} \approx \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \pmod q,$$

We wish to convert $\mathbf{c}_{\text{in}}$ to an $n\mathbf{c}_{\text{out}}$-dimensional ciphertext $\mathbf{c}_{\text{out}}$ that still encrypts μ, but with respect to some possibly different secret key $\mathbf{s}_{\text{out}}$. The first main insight is that

$$\langle \mathbf{s}_{\text{in}} \otimes \mathbf{c}_{\text{in}} \rangle = \mathbf{s}_{\text{in}}^t \cdot \mathbf{c}_{\text{in}} = (\mathbf{s}_{\text{in}}^t \mathbf{G}) \cdot \mathbf{G}^{-1}(\mathbf{c}_{\text{in}}) \approx \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \pmod q, \tag{1.2}$$

where $\mathbf{G}$ is a *gadget* matrix with n_{in} rows; $\mathbf{G}^{-1}(\mathbf{c}_{\text{in}})$ is a *short* integer vector. Key-switching is made possible by publishing a suitable "encryption" of $\mathbf{s}_{\text{in}}$ under $\mathbf{s}_{\text{out}}$, namely, a matrix $\mathbf{K}$ over $\mathbb{Z}_q$ such that

$$\mathbf{s}_{\text{in}}^t \mathbf{K} \approx \mathbf{s}_{\text{out}}^t \mathbf{G} \pmod q, \tag{1.3}$$

where the approximation hides small errors. Essentially, the columns of $\mathbf{K}$ are LWE samples with respect to $\mathbf{s}_{\text{out}}$, with $\mathbf{s}_{\text{in}}^t \mathbf{G}$ added to the last row. Assuming the hardness of LWE, it is easy to prove that such a $\mathbf{K}$ is pseudorandom and hence safe to publish, as long as $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ are independent[1].

To key-switch the ciphertext $\mathbf{c}_{\text{in}}$ using $\mathbf{K}$, we simply output $\mathbf{c}_{\text{out}} = \mathbf{K} \cdot \mathbf{G}^{-1}(\mathbf{c}_{\text{in}})$. Combining this with Equations 1.2 and 1.3, we see that

$$\mathbf{s}_{\text{out}}^t \cdot \mathbf{c}_{\text{out}} = (\mathbf{s}_{\text{out}}^t \mathbf{K}) \cdot \mathbf{G}^{-1}(\mathbf{c}_{\text{in}}) \approx (\mathbf{s}_{\text{in}}^t \mathbf{G}) \cdot \mathbf{G}^{-1}(\mathbf{c}_{\text{in}}) \approx \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \pmod q,$$

where the first approximation holds by Equation 1.3 and because $\mathbf{G}^{-1}(\mathbf{c}_{\text{in}})$ is a *short* integer vector. Therefore, $\mathbf{c}_{\text{out}}$ encrypts μ under $\mathbf{s}_{\text{out}}$, as desired.

Error management and modulus switching. Recall from above that homomorphic addition and multiplication increase the magnitude of the error terms in the resulting ciphertexts; in particular, the error in a homomorphic product of ciphertexts is the product of the individual errors (plus a little more, due to key-switching). This severely limits the homomorphic capacity and the hardness/efficiency tradeoff of the scheme: the modulus q must be larger than the error in the final ciphertext, so freshly encrypted ciphertexts must have very small error rate relative to q. More specifically, the scheme as described so far can homomorphically evaluate circuits of only a fixed *logarithmic* depth $d = \mathcal{O}(\log \lambda)$ in the security parameter λ, because the modulus must be $q = \lambda^{\Omega(2^d)}$.

A very simple but powerful modulus reduction technique, first used in [BV11a] and then exploited to its full potential in [BGV12], greatly extends the homomorphic capacity to circuits of any a-priori bounded *polynomial* depth $d = \text{poly}(\lambda)$ in the security parameter. The idea is that by strategically scaling down the modulus by some $\text{poly}(n)$ factor (typically, before homomorphic multiplication), we can decrease the *absolute* error $|e|$ to some small fixed polynomial, even though the relative error *rate* $|e|/q$ remains essentially unchanged. Because the absolute error is what determines the error growth in homomorphic multiplication, modulus reduction yields an arbitrage that allows us to evaluate a depth-d circuit with an (original) modulus of only $q = \lambda^{\Omega(d)}$. More specifically, after evaluating d layers of a circuit, our original modulus q shrinks to $q/n^{\mathcal{O}(d)}$ while the absolute error

[1] Note that if we want to switch from key $\mathbf{s}_{\text{in}} = \mathbf{s} \otimes \mathbf{s}$ back to the *original* $\mathbf{s}_{\text{out}} = \mathbf{s}$, then the keys are not independent. In such a case we can simply make the "circular security" assumption the publishing $\mathbf{K}$ is safe, though this assumption is still not very well understood.

remains $\mathsf{poly}(n)$. So it suffices to set the original modulus as $q = n^{\Theta(d)}$ is order to ensure correct decryption after a depth-d computation.

The modulus reduction technique relies on the normal from of the LWE problem, where the secret $\mathbf{s} \in \mathbb{Z}^n$ is a rather short integer vector drawn from the error distribution. The main idea is that *rounding* an LWE sample (having a short secret) from $\mathbb{Z}_q$ to $\mathbb{Z}_{q'}$ scales the absolute error by a q'/q factor, plus a small additive term:

$$\langle \mathbf{s}, \mathbf{c} \rangle \in e + q\mathbb{Z} \implies \langle \mathbf{s}, \lfloor \mathbf{c} \rceil_{q'} \rangle \in \langle \mathbf{s}, \frac{q'}{q} \cdot \mathbf{c} + [-\frac{1}{2}, \frac{1}{2}]^n \rangle \subseteq (\frac{q'}{q} \cdot \mathbf{e} + \|\mathbf{s}\|\sqrt{n} \cdot [-\frac{1}{2}, \frac{1}{2}] + q'\mathbb{Z}).$$

In the FHE context, one can verify that the above rounding also preserves the encrypted message, when the ciphertext is in most-significant bit form (i.e. $\langle \mathbf{s}, \mathbf{c} \rangle \approx \mu \cdot \lfloor \frac{q}{2} \rceil \pmod q$).

We also mention that Brakerski [Bra12] gave an alternative "scale invariant" method of homomorphic multiplication that increases the error rate by only a fixed $\mathsf{poly}(n)$ factor, regardless of the absolute error of the input ciphertexts. Using this method, modulus reduction becomes optimal. However, it can still be useful because the ciphertext sizes and computation times become smaller as the modulus shrinks.

Bootstrapping. Even with all of the above techniques, homomorphic operations still always increase the error *rate* of a ciphertext, by as much as a polynomial factor per operation. Therefore, the schemes described so far can only homomorphically evaluate circuits of an a-priori bounded depth; such systems are frequently called "somewhat homomorphic" or "leveled" (we ignore the precise technical distinction between these terms).

A beautiful idea from Gentry's original work [Gen09], called *bootstrapping* or sometimes *refreshing*, makes it possible to reduce the error *rate* of a ciphertext, thus enabling unbounded homomorphic computation. Suppose we have a ciphertext c that encrypts some (unknown) message μ under a secret key s, where the error rate of c is too large to perform further homomorphic operations on it. The idea behind boostrapping is to *homomorphically evaluae the decryption function* on a low-error encryption $c_s = \mathsf{Enc}(s)$ of the secret key s, which is included as part of the public key. More specifically, we homomorphically evaluate the function $f_c(\cdot) = \mathsf{Dec}(\cdot, c)$ on the ciphertext c_s. (Note that the ciphertext c to be refreshed is "hard-coded" into the function $f_c(\cdot)$, whereas the secret key is treated as the function argument.) Because $f_c(s) = \mathsf{Dec}(s, c) = \mu$, it follows that homomorphic evaluation of f_c on as *encryption* of s (namely, c_s) produces an *encryption* of μ. Moreover, as long as the circuit depth of f_c and the error rate of c_s are small enough, the error rate of the output ciphertext will be substantially smaller than that of the input ciphertext c, as desired. In particular, decryption can be performed in $\mathcal{O}(\log \lambda)$ depth, so it suffices for c_s to have some $\lambda^{-\mathcal{O}(\log \lambda)}$ error rate.

Because bootstrapping involves the homomorphic evaluation of a somewhat complex function, it is not very efficient (see for example [GH11b]). However, boostrapping has been intensively studied and improved in various ways [GH11a; BGV12], culminating in ring-LWE-based methods that run in only polylogarithmic $\tilde{\mathcal{O}}(1)$ time per encrypted bit [GHS12a; AP13], where the hidden $\log^{\mathcal{O}(1)}(\lambda)$ factors are not exceedingly large.

We conclude this discussion by noting that, in order to yield *unbounded* homomorphic operations, boostrapping requires a "circular" encryption of the secret key *under itself*. It is unknown whether it is provably secure (under a standard assumption) to reveal such an encryption, but no attack expoiting such a circular encryption is known. So to date, all unbounded FHE schemes require an additional assumption of circular security for the secret key.

1.3. Contributions of this Thesis

This thesis studies lattice-based cryptography and especially its Gaussian sampling component. It brings in at once new theoretical results, new constructions and new practical improvements. These results mainly come from: a candidate to the NIST post-quantum cryptography standardization process; two published papers; and two not yet published papers.

1.3.1. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU [FHK+17]

Falcon is a cryptographic signature algorithm submitted to NIST Post-Quantum Cryptography Project with P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, G. Seiler, W. Whyte and Z. Zhang.

The point of a post-quantum cryptographic algorithm is to keep on ensuring its security characteristics even faced with quantum computers. Quantum computers are deemed feasible, according to our current understanding of the laws of physics, but some significant technological issues remain to be solved in order to build a fully operational unit. Such a quantum computer would very efficiently break the usual asymmetric encryption and digital signature algorithms based on number theory (RSA, DSA, Diffie-Hellman, ElGamal, and their elliptic curve variants). Falcon is based on the theoretical framework of Gentry, Peikert and Vaikuntanathan for lattice-based signature schemes. We instantiate that framework over NTRU lattices, with a trapdoor sampler called "fast Fourier sampling". The underlying hard problem is the short integer solution problem (SIS) over NTRU lattices, for which no efficient solving algorithm is currently known in the general case, even with the help of quantum computers.

Falcon offers the following features:

- **Security:** a true Gaussian sampler is used internally, which guarantees negligible leakage of information on the secret key up to a practically infinite number of signatures (say 2^{64}).
- **Compactness:** thanks to the use of NTRU lattices, signatures are substantially shorter than in any lattice-based signature scheme with the same security guarantees, while the public keys are around the same size.
- **Speed:** use of fast Fourier sampling allows for very fast implementations, in the thousands of signatures per second on a common computer; verification is five to ten times faster.
- **Scalability:** operations have cost $\mathcal{O}(n \log n)$ for degree n, allowing the use of very long-term security parameters at moderate cost.
- **RAM Economy:** the enhanced key generation algorithm of Falcon uses less than 30 kilobytes of RAM, a hundredfold improvement over previous designs such as NTRUSign. Falcon is compatible with small, memory-constrained embedded devices.

1.3.2. Sampling From Arbitrary Centered Discrete Gaussians For Lattice-based Cryptography [AAR17]

This publication is cosigned with C. Aguilar-Melchor and M. R. Albrecht and published at the 15th International Conference on Applied Cryptography and Network Security (ACNS) 2017.

Non-Centered Discrete Gaussian sampling is a fundamental building block in many lattice-based constructions in cryptography, such as signature and identity-based encryption schemes. On the

one hand, approaches that rely on center-dependent precomputation, e.g. cumulative distribution tables (CDT), Knuth-Yao, the alias method, discrete Zigurat and their variants, are the fastest known algorithms to sample from a discrete Gaussian distribution. However, center-dependent precomputation must be done for many potential centers in $[0, 1)$ in order to be usable by these algoritms, making them impracticable for non-centered discrete Gaussian sampling. On the other hand, rejection sampling allows to sample from a discrete Gaussian distribution for all real centers without prohibitive precomputation costs but needs costly floating-point arithmetic and several trials per sample.

In this work, we study how to reduce the number of centers for which we have to precompute tables and propose a non-centered CDT algorithm with practicable size of precomputed tables as fast as its centered variant. Finally, we provide some experimental results for our open-source C++ implementation indicating that our sampler increases the rate of Peikert's algorithm for sampling from arbitrary lattices by a factor 3 with precomputation storage up to 6.2 MB.

1.3.3. CDT-based Gaussian Sampling: From Multi to Double Precision [AR18]

This publication is cosigned with C. Aguilar-Melchor and published in the IEEE Transactions on Computers (TC) journal.

The Rényi divergence is a measure of closeness of two probability distributions which has found several applications over the last years as an alternative to the statistical distance in lattice-based cryptography. A tight bound has recently been presented for the Rényi divergence of distributions that have a bounded relative error. We show that it can be used to bound the precision requirement in Gaussian sampling to the IEEE 754 floating-point standard double precision for usual lattice-based signature parameters by using a modified cumulative distribution table (CDT), which reduces the memory needed by CDT-based algorithms and, makes their constant-time implementation faster and simpler.

Then, we apply this approach to a variable-center variant of the CDT algorithm which occasionally requires the online computation of the cumulative distribution function. As a result, the amount of costly floating-point operations is drastically decreased, which makes the constant-time and cache-resistant variants of this algorithm viable and efficient. Finally, we provide some experimental results indicating that comparing to rejection sampling our approach increases the GPV signature rate by a factor 4 to 8 depending on the security parameter.

1.3.4. Delegating Elliptic-Curve Operations with Homommorphic Encryption [ADG+18]

This publication is cosigned with C. Aguilar-Melchor, J.-C. Deneuville, P. Gaborit and T. Lepoint and published in the IEEE Workshop on Security and Privacy in the Cloud (SPC) 2018.

The landscape of Fully Homomorphic Encryption (FHE) has known great changes these past years. As computational costs drop, libraries are developed and new applications become possible. Prototypes demonstrating private health diagnosis, signal processing, genome statistics and database queries plus the recent NIST's call for post-quantum proposals spur hope on the practical deployment of FHE in the near future. However, in most of these applications, increasing the privacy, security or enabling new functionalities comes in pair with some significant computation and/or communication burden.

In this work, we depart from the latter paradigm and demonstrate that FHE might be useful to increase the *practical performance* of elliptic-curve cryptography. More precisely, we show how

to reduce the computational burden of computing elliptic-curve points from a generator through delegation. We show that using homomorphic encryption it is possible to reduce in practice computational costs even with respect to traditional, not based on homomorphic-encryption, delegation protocols. We demonstrate the feasibility of our protocols with proof-of-concept implementations using HElib (adapted to handle multiprecision plaintext space moduli).

References

[AAR17] Carlos Aguilar-Melchor, Martin R. Albrecht, and Thomas Ricosset. *Sampling from Arbitrary Centered Discrete Gaussians for Lattice-Based Cryptography*. In: *ACNS 17*. Ed. by Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi. Vol. 10355. LNCS. Springer, Heidelberg, July 2017, pp. 3–19 (cit. on p. 18).

[ADG+18] Carlos Aguilar-Melchor, Jean-Christophe Deneuville, Philippe Gaborit, Tancrede Lepoint, and Thomas Ricosset. "Delegating Elliptic-Curve Operations with Homomorphic Encryption". In: *4th IEEE Workshop on Security and Privacy in the Cloud* (2018) (cit. on p. 19).

[Ajt96] Miklós Ajtai. *Generating Hard Instances of Lattice Problems (Extended Abstract)*. In: *28th ACM STOC*. ACM Press, May 1996, pp. 99–108 (cit. on pp. 4, 7–9).

[AP13] Jacob Alperin-Sheriff and Chris Peikert. *Practical Bootstrapping in Quasilinear Time*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–20 (cit. on p. 17).

[AR18] Carlos Aguilar-Melchor and Thomas Ricosset. "CDT-based Gaussian Sampling: From Multi to Double Precision". In: *IEEE Trans. Computers* (2018) (cit. on p. 19).

[Bab86] L. Babai. "On Lovász' lattice reduction and the nearest lattice point problem". In: *Combinatorica* 6.1 (Mar. 1986), pp. 1–13 (cit. on pp. 13, 14, 64).

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. *Random Oracles in a Quantum World*. In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 41–69 (cit. on pp. 13, 63, 64, 70).

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. *Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 533–556 (cit. on p. 9).

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping*. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on pp. 14, 16, 17, 114, 115, 117, 121).

[Bra12] Zvika Brakerski. *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 868–886 (cit. on pp. 14, 17).

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. In: *52nd FOCS*. Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 97–106 (cit. on pp. 14, 16).

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. *Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages*. In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 505–524 (cit. on p. 14).

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. *Batch Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 315–335 (cit. on pp. 14, 117).

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. *Fully Homomorphic Encryption over the Integers with Shorter Public Keys*. In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 487–504 (cit. on p. 14).

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. *Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 446–464 (cit. on p. 14).

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. *Lattice Signatures and Bimodal Gaussians*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 40–56 (cit. on pp. 13, 27, 32, 53).

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43 (cit. on p. 14).

[DN12b] Léo Ducas and Phong Q. Nguyen. *Learning a Zonotope and More: Cryptanalysis of NTRUSign Countermeasures*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 433–450 (cit. on pp. xiii, xv, 8, 62, 64, 68).

[FHK+17] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU*. 2017. URL: `https://falcon-sign.info` (cit. on pp. 18, 61).

[FS87] Amos Fiat and Adi Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*. In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194 (cit. on p. 13).

[Gen09] Craig Gentry. *Fully homomorphic encryption using ideal lattices*. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on pp. 9, 14, 17, 114, 117).

[Gen10] Craig Gentry. *Toward Basing Fully Homomorphic Encryption on Worst-Case Hardness*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 116–137 (cit. on p. 14).

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. *Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits*. In: *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. FOCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 40–49 (cit. on p. 9).

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. *Public-Key Cryptosystems from Lattice Reduction Problems*. In: *CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 112–131 (cit. on pp. xiii, 7–9, 13, 62, 64).

[GH11a] Craig Gentry and Shai Halevi. *Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits*. In: *52nd FOCS*. Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 107–109 (cit. on p. 17).

[GH11b] Craig Gentry and Shai Halevi. *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 129–148 (cit. on p. 17).

[GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. *Ring Switching in BGV-Style Homomorphic Encryption*. In: *SCN 12*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. LNCS. Springer, Heidelberg, Sept. 2012, pp. 19–37 (cit. on p. 14).

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Better Bootstrapping in Fully Homomorphic Encryption*. In: *PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. LNCS. Springer, Heidelberg, May 2012, pp. 1–16 (cit. on p. 17).

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Fully Homomorphic Encryption with Polylog Overhead*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 465–482 (cit. on p. 14).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. *Trapdoors for hard lattices and new cryptographic constructions*. In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206 (cit. on pp. xii, xiii, 9, 13, 14, 31, 62–65).

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92 (cit. on p. 14).

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. *Attribute-based encryption for circuits*. In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 545–554 (cit. on p. 9).

[HHP+03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. *NTRUSIGN: Digital Signatures Using the NTRU Lattice*. In: *CT-RSA 2003*. Ed. by Marc Joye. Vol. 2612. LNCS. Springer, Heidelberg, Apr. 2003, pp. 122–140 (cit. on pp. xiii, 8, 62, 64).

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A Ring-Based Public Key Cryptosystem*. In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. 1998, pp. 267–288 (cit. on pp. 7, 10, 11, 54, 65).

[Kle00] Philip N. Klein. *Finding the closest lattice vector when it's unusually close*. In: *11th SODA*. Ed. by David B. Shmoys. ACM-SIAM, Jan. 2000, pp. 937–941 (cit. on pp. xiv, 13, 63, 64, 67).

[KLS17] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. *A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model*. Cryptology ePrint Archive, Report 2017/916. `http://eprint.iacr.org/2017/916`. 2017 (cit. on pp. 13, 70).

[LLL82] A.K. Lenstra, H.W.jun. Lenstra, and László Lovász. "Factoring polynomials with rational coefficients". In: *Mathematische Annalen* 261.4 (1982), pp. 515–534 (cit. on p. 4).

[LM08] Vadim Lyubashevsky and Daniele Micciancio. *Asymptotically Efficient Lattice-Based Digital Signatures*. In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Heidelberg, Mar. 2008, pp. 37–54 (cit. on p. 13).

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors over Rings*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 1–23 (cit. on pp. 10, 11).

[Lyu08] Vadim Lyubashevsky. *Lattice-Based Identification Schemes Secure Under Active Attacks*. In: *PKC 2008*. Ed. by Ronald Cramer. Vol. 4939. LNCS. Springer, Heidelberg, Mar. 2008, pp. 162–179 (cit. on p. 13).

[Lyu09] Vadim Lyubashevsky. *Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures*. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 598–616 (cit. on p. 13).

[Lyu12] Vadim Lyubashevsky. *Lattice Signatures without Trapdoors*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 738–755 (cit. on p. 13).

[McE78] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *Deep Space Network Progress Report* 44 (Jan. 1978), pp. 114–116 (cit. on p. 7).

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. *Additively Homomorphic Encryption with d-Operand Multiplications*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 138–154 (cit. on p. 15).

[Mic01] Daniele Micciancio. *Improving Lattice based cryptosystems using the Hermite Normal Form*. In: *Cryptography and Lattices Conference — CaLC 2001*. Vol. 2146. Lecture Notes in Computer Science. Springer-Verlag, 2001, pp. 126–145 (cit. on p. 8).

[Mic02] Daniele Micciancio. *Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions from Worst-Case Complexity Assumptions*. In: *43rd FOCS*. IEEE Computer Society Press, Nov. 2002, pp. 356–365 (cit. on p. 10).

[MR04] Daniele Micciancio and Oded Regev. *Worst-Case to Average-Case Reductions Based on Gaussian Measures*. In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 372–381 (cit. on p. 12).

[MR07] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM J. Comput.* 37.1 (Apr. 2007), pp. 267–302 (cit. on pp. 12, 31, 33).

[Ngu99] Phong Q. Nguyen. *Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto'97*. In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 288–304 (cit. on p. 7).

[NR06] Phong Q. Nguyen and Oded Regev. *Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 271–288 (cit. on pp. xiii, xv, 7, 8, 13, 62, 64, 68).

[Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238 (cit. on p. 13).

[Pei10] Chris Peikert. *An Efficient and Parallel Gaussian Sampler for Lattices*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 80–97 (cit. on pp. xiv, xix, xxi, 14, 27, 33, 40, 41, 43, 53, 63, 67).

[Pei16] Chris Peikert. "A Decade of Lattice Cryptography". In: *Foundations and Trends in Theoretical Computer Science* 10.4 (2016), pp. 283–424 (cit. on pp. 3, 10).

[PLP16] Rafaël del Pino, Vadim Lyubashevsky, and David Pointcheval. *The Whole is Less Than the Sum of Its Parts: Constructing More Efficient Lattice-Based AKEs*. In: *SCN 16*. Ed. by Vassilis Zikas and Roberto De Prisco. Vol. 9841. LNCS. Springer, Heidelberg, Aug. 2016, pp. 273–291 (cit. on pp. 13, 63, 64).

[Rab79] M. O. Rabin. *igitalized signatures and public-key functions as intractable as factorization*. Tech. rep. Cambridge, MA, USA, 1979 (cit. on p. 13).

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179 (cit. on pp. 9, 14).

[Reg05] Oded Regev. *On lattices, learning with errors, random linear codes, and cryptography*. In: *37th ACM STOC*. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93 (cit. on p. 10).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126 (cit. on p. 13).

[Sha84] Adi Shamir. "A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem". In: *IEEE Trans. Information Theory* 30.5 (1984), pp. 699–704 (cit. on p. 4).

[SS11] Damien Stehlé and Ron Steinfeld. *Making NTRU as Secure as Worst-Case Problems over Ideal Lattices*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 27–47 (cit. on pp. xiii, 7, 62, 65, 66).

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations". In: *Designs, Codes and Cryptography* 71.1 (Apr. 2014), pp. 57–81 (cit. on pp. 14, 117, 118).

[Unr17] Dominique Unruh. "Post-Quantum Security of Fiat-Shamir". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 398. URL: `http://eprint.iacr.org/2017/398` (cit. on pp. 13, 70).

Simple, easy-to-understand algorithms will survive longer, all other things being roughly equal. Unfortunately, such algorithms are usually slower than their more sophisticated counterparts. The notion of time itself is of course relative.

Non-Uniform Random Variate Generation – Luc Devroye

Gaussian Sampling over the Integers for Lattice Trapdoors 2

Gaussian sampling over the integers is a core building block for many lattice-based cryptographic schemes, and in particular for which using a GPV-like trapdoor, as discussed in section 1.2.2.1. The GPV signature framework is quite demanding when it comes to Gaussian sampling over the integers. The Gaussians from which we need to sample have very small standard deviation and their center cannot be determined before the signature algorithm is executed. Indeed, most of the existing Gaussian sampling algorithms work only for fixed center (e.g. Inversion Sampling with a cumulative distribution table (CDT) [Pei10], Knuth-Yao [DG14], Discrete Ziggurat [BCG+14], Bernoulli Sampling [DDLL13]) or do not work for small standard deviation (Convoluted Sampling [MW17]). Before the techniques developed in this chapter, the only samplers which fit the GPV use case was the rejection methods (e.g. the Karney's sampler [Kar16]), which do not use precomputed data, but need costly arithmetic and several trials per sample.

In this chapter, after some preliminaries, in section 2.1, about non-uniform random number generation and discrete gaussian distributions, we present, in section 2.2, a study to limit the number of centers for which we have to precompute CDTs in the variable-center case, which leads to a variable-center CDT algorithm with practicable size of precomputation as fast as its fixed-center variant. Then, in section 2.3, we show that using the Rényi divergence, which is a measure of closeness of two probability distributions, we can bound the precision requirement in CDT-based Gaussian sampling to the IEEE 754 floating-point standard double precision for usual lattice-based signature parameters. Finally, in section 2.4, we present techniques to protect implementations against timing-attacks, and provide performances indicating that our approach outperforms by a factor of up to 75 times the fastest known rejection method (i.e. the Karney's sampler).

Contents

2.1. Preliminaries

In this section, we present some preliminaries about the theory of non-uniform random variate generation [Dev86], and useful tools in cryptography, about floating-point arithmetic and statistical closeness measures, to ensure the accuracy of these algorithms. Then we discute some properties of the discrete Gaussian distributions, and present the known methods to sample according to them.

2.1.1. Non-Uniform Random Number Generation

Non-uniform random number generation is about to generate sequences of integers with certain non-uniform distributions, given that a perfect uniform random number generator is available. The assumption that a perfect uniform random number generator is available is now quite unrealistic, but, in cryptography this assumption is usually replaced by the use of a pseudo-random number generator (PRNG) which has been designed to be cryptographically secure (e.g. Salsa20 [Ber08]). This uniform (pseudo-)random number generator is our fundamental building block, we assume that the random sequence of bits obtained from it, separated in words $U_1, U_2, \ldots$, is uniform, i.e. the uniform random generator produces a sequence $U_1, U_2, \ldots$ with $\forall i \in \mathbb{N}^*, U_i \in \{0,1\}^\ell$ of independent random variables with a uniform distribution on $\{0,1\}^\ell$.

A (non-uniform) random number generator is a program that halts with probability one and exits with an integer X. This X is called a *random variate*. Because of our theoretical assumption we can treat random variate as if they were random variables. Note also that if we can produce one random variate X, then we are able to produce a sequence $X_1, X_2, \ldots$ of independent random variates distributed as X (this follows from our assumption that an uniform random number generator is available). This facilitates our task a lot: rather than having to concentrate on infinite sequences, we just need to look at the properties of single random variates.

For theoretical purposes, it is necessary to equate time with the number of "fundamental" operations performed before the algorithm halts, this is called *time complexity* of the algorithm. This leads to our second assumption: The fundamental operations in our computer include addition, multiplication, division, compare, truncate, move, generate a uniform random variate and exp. (This does not implies that each of these operations takes one unit of time, the size of the operand(s) has to be considered.) The time complexity of an algorithm, denoted T, is the time required to produce one random variate. In many (non-constant-time) cases, T itself is a random variable since it is a function of $U_1, U_2, \ldots$. We note that we are mainly interested in generating independent sequences of random variables. The average time complexity per random variate in a sequence of length n is

$$\frac{1}{n}\sum_{i=1}^{n} T_i$$

where T_i is the complexity for the i-th random variate. By the strong law of large numbers, we know that this average tends with probability one to the expected complexity, $E(T)$.

2.1.1.1. Floating-Point Arithmetic

Since computers are finite memory machines, they cannot store real numbers, let alone generate random variables with a given density. This led us to assume that our computer can store and manipulate real numbers. Obviously, in practice computers can use floating-point arithmetic (FPA) to deal with this. Floating-point arithmetic is arithmetic using formulaic representation of real numbers as an approximation so as to support a trade-off between range and precision. A real number

is, following the IEEE 754 double-precision binary floating-point (binary64) format, represented approximately in FPA to a mantissa $m \in (-2,-1] \cup [1,2)$ of precision $p = 53$ bits such that $|m| \cdot 2^{52} \in [2^{52}, 2^{53}-1] \cap \mathbb{Z}$, scaled using an 11-bit (biased) exponent $e \in [-1022, 1023] \cap \mathbb{Z}$ in base two. Such that any floating-point number $\bar{x} \in \mathbb{FP}_p$ (i.e. any number that can be represented exactly as a floating-point number given a precision p) is of the form $\bar{x} = m \cdot 2^e$. It is interesting to note that, in the IEEE 754 binary64 format, the most significant bit of $|m|$ is always equal to one, therefore it is not stored, the mantissa is represented to a bit sign and the 52 bits of its fractional part.

Relative error. In floating-point arithmetic the relative error of a number $\bar{x}$, which is the floating-point approximation with a p-bit mantissa of a real number x, is defined as $\delta_{\mathsf{RE}}(x,\bar{x}) := |x-\bar{x}|/|x|$. The IEEE 754 specification on denormalized numbers guarantees that the relative error is bounded by $\delta_{\mathsf{RE}}(x,\bar{x}) \leq 2^{1-p}$ for any elementary arithmetic operation (addition, substraction, multiplication, division, square root and fused multiply-add) and numeric libraries compute usually also the basic transcendental functions with $\delta_{\mathsf{RE}}(x,\bar{x}) \leq 2^{2-p}$. We recall also the useful extended notion of relative error, from [MW17], to any two distributions P and Q with the same support $S := \mathrm{Supp}(P) = \mathrm{Supp}(Q)$:

$$\delta_{\mathsf{RE}}(P,Q) := \max_{x \in S} \delta_{\mathsf{RE}}(P(x),Q(x)) = \max_{x \in S} \frac{|P(x)-Q(x)|}{P(x)}.$$

2.1.1.2. Statistical Closeness Measures

With the assumptions given above, we can demand that the random sequence obtained by a non-uniform random number generator has the exact distribution that was asked. Algorithms or generators with this property is called exact. Exact algorithms approach realty, for non-transcendental densities, if we use extended precision arithmetic (e.g. MPFR [FHL+07]). In this manuscript, for the sake of efficiency, we are interested about inexact algorithms, which are usually algorithms that are based upon a mathematical approximation. However, in cryptography, it is preferable to have some security guarantees, therefore we use a measure of closeness to determine the precision needed by practical implementations of non-uniform random number generators.

The well known statistical distance and the Rényi divergence are two measures of closeness of two probability distributions. The Rényi divergence has found several applications over the last years as an alternative to the statistical distance in lattice-based cryptography.

Statistical distance. The statistical distance (SD for short) between two discrete probability distributions P and Q, with the same support $S := \mathrm{Supp}(P) = \mathrm{Supp}(Q)$, is defined as:

$$\Delta(P,Q) := \frac{1}{2} \sum_{x \in S} |P(x) - Q(x)|$$

Rényi divergence. We recall the definition from [BLL+15] of the Rényi divergence (RD for short): For any two discrete probability distributions P and Q such that $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$ and $a \in (1, +\infty)$, we define the Rényi divergence of order a by

$$R_a(P\|Q) := \left(\sum_{x \in \mathrm{Supp}(P)} \frac{P(x)^a}{Q(x)^{a-1}} \right)^{\frac{1}{a-1}}$$

We define the Rényi divergences of orders 1 and $+\infty$ by

$$R_1(P\|Q) := \exp\left(\sum_{x\in\mathrm{Supp}(P)} P(x)\log\frac{P(x)}{Q(x)}\right)$$

and

$$R_\infty(P\|Q) := \max_{x\in\mathrm{Supp}(P)} \frac{P(x)}{Q(x)}$$

The Rényi divergence is an alternative to the statistical distance as measure of distribution closeness, where we replace the difference in the statistical distance, by the ratio in the Rényi divergence. Note that the Rényi divergence of order 1 is know as (the exponential of) the Kullback-Leibler divergence (KLD for short). Remark that the Rényi divergence is not a distance. We recall also some important properties of the Rényi divergence from [BLL+15; EH14]:

Lemma 2.1 ([BLL+15, Lemma 2.7])**.** *Let $a \in [1,+\infty]$. Let P and Q denote distributions with $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$. Then the following properties hold:*

- ***Data Processing Inequality:*** *For any function f, where P^f (resp. Q^f) denotes the distributions of $f(y)$ induced by sampling y from P (resp. Q), we have:*

$$R_a(P^f\|Q^f) \le R_a(P\|Q).$$

- ***Multiplicativity:*** *Assume P and Q are the two distributions of a pair of random variables (Y_1,Y_2). For $i\in\{1,2\}$, let P_i (resp. Q_i) denote the marginal distribution of Y_i under P (resp. Q). Then, if Y_1 and Y_2 are independent, we have:*

$$R_a(P\|Q) = R_a(P_1\|Q_1)R_a(P_2\|Q_2).$$

- ***Probability Preservation:*** *Let $A\subseteq\mathrm{Supp}(Q)$ be an arbitrary event. If $a\in(1,+\infty)$, then:*

$$Q(A) \ge P(A)^{\frac{a}{a-1}}/R_a(P\|Q),$$
$$Q(A) \ge P(A)/R_\infty(P\|Q).$$

It is worth noting that the statistical distance $\Delta(P,Q) \le \frac{1}{2}\delta_{\mathsf{RE}}(P,Q)$.

Lemma 2.2 ([PDG14, Lemma 2] strengthened in [MW17, Lemma 2.1])**.** *Let P and Q be two distributions with the same support, if $\delta_{RE}(P,Q)\le 1/4$, then:*

$$R_1(P\|Q) \le \exp\left(\delta_{RE}(P,Q)^2\right).$$

Lemma 2.3 ([Pre17, Lemma 3])**.** *Let P and Q be two distributions with the same support, then for $a\in(1,+\infty)$:*

$$R_a(P\|Q) \le \left(1+\frac{a(a-1)\delta_{RE}(P,Q)^2}{2(1-\delta_{RE}(P,Q))^{a+1}}\right)^{\frac{1}{a-1}}.$$

2.1.2. Gaussian Sampling over the Integers

Throughout this chapter, we denote the set of real numbers by $\mathbb{R}$ and the integers by $\mathbb{Z}$. We extend any real function $f()$ to a countable set A by defining $f(A) = \sum_{x \in A} f(x)$. For any distribution D, we note $\mathrm{Supp}(D)$ its support and we denote by U_I the uniform distribution on I. We note λ the security parameter of a cryptographic scheme and q_s the number of queries that an attacker can make to the private-key functionality oracle.

The discrete Gaussian distribution on $\mathbb{Z}$ is defined as the probability distribution whose unnormalized density function is

$$\begin{aligned} \rho : \quad & \mathbb{Z} \to [0,1] \\ & x \to e^{\frac{-x^2}{2}} \end{aligned}$$

If $s \in \mathbb{R}_+^*$ and $c \in \mathbb{R}$, then we extend this definition to

$$\rho_{s,c}(x) := \rho\left(\frac{x-c}{s}\right).$$

and denote $\rho_{s,0}(x)$ by $\rho_s(x)$. For any mean $c \in \mathbb{R}$ and parameter $s \in \mathbb{R}_+^*$ we can now define the discrete Gaussian distribution $D_{s,c}$, for any integer x, as

$$D_{s,c}(x) := \frac{\rho_{s,c}(x)}{\rho_{s,c}(\mathbb{Z})}.$$

Note that the standard deviation of this distribution is $\sigma = s/\sqrt{2\pi}$. We also define $\mathrm{cdf}_{s,c}$ as the cumulative distribution function (cdf) of $D_{s,c}$

$$\mathrm{cdf}_{s,c}(x) := \sum_{i=-\infty}^{x} D_{s,c}(i).$$

Gaussian measure. An interesting property of discrete Gaussian distributions with a parameter s greater than the smoothing parameter (see section 1.2.1) is that the Gaussian measure, i.e. $\rho_{s,c}(\mathbb{Z})$ for $D_{s,c}$, is essentially the same for all centers.

Lemma 2.4 (From the proof of [MR07, Lemma 4.4]). *For any $\epsilon \in (0,1)$, $s > \eta_\epsilon(\mathbb{Z})$ and $c \in \mathbb{R}$ we have:*

$$\frac{\rho_{s,0}(\mathbb{Z})}{\rho_{s,c}(\mathbb{Z})} \in \left[1, \frac{1+\epsilon}{1-\epsilon}\right]$$

Tailcut parameter. To deal with the infinite domain of Gaussian distributions, algorithms usually take advantage of their rapid decay to sample from a finite domain. The next lemma is useful in determining the tailcut parameter t.

Lemma 2.5 ([GPV08, Lemma 4.2]). *For any $\epsilon > 0$, $s > \eta_\epsilon(\mathbb{Z})$ and $t > 0$, we have:*

$$E_{\textit{tailcut}} := \Pr_{X \sim D_{s,c}}[|X - c| > ts] < 2e^{-\pi t^2}\frac{1+\epsilon}{1-\epsilon}.$$

In particular, Lemma 2.5 implies that $\mathsf{E}_{\mathsf{tailcut}} < 2^{-\lambda}$ for $t = \sqrt{\lambda}/2$, $\epsilon \leq 1/3$ and $\lambda \geq 12$.

Gaussian sampling over the integers is a core building block for many lattice-based cryptographic schemes. As a consequence, several algorithms for performing this operation have been proposed

in the recent years. However, while indifferentiability from an ideal Gaussian, speed, memory-efficiency, conceptual simplicity, genericity and resistance against side-channel attacks all seem important goals to reach, it has proved difficult for the algorithms proposed to achieve all these properties simultaneously.

On the one hand, center-dependent samplers are the fastest known to sample from a discrete Gaussian distribution. However, they use a relatively large precomputed table for each possible real center, w.l.o.g. in $[0, 1)$, making them impracticable for variable-center distributions, as used in lattice trapdoors (e.g. in FALCON). On the other hand, rejection methods allow to sample from a discrete Gaussian distribution for all real centers without prohibitive precomputation cost but needs costly arithmetic and several trials per sample.

2.1.2.1. Rejection Methods

Straightforward rejection sampling [Von51] is a classical method to sample from any distribution by sampling from a uniform distribution and accept the value with a probability equal to its probability in the target distribution. It does not use precomputed data but needs floating-point arithmetic and several trials by sample. A variant of this straightforward rejection sampling approach use "lazy" floating-point computations [DN12a] with IEEE 754 standard double precision floating-point numbers in most cases.

Bernoulli sampling [DDLL13] introduces an exponential bias from Bernoulli variables, which can be efficiently sampled specially in circuits. The bias is then corrected in a rejection phase based on another Bernoulli variable. This approach is particularly suited for embedded devices for the simplicity of the computation, the small center-dependent precomputation, and the near-optimal entropy consumption.

Currently the fastest rejection approach for arbitrary variable center is the Karney sampler [Kar16] which does not use floating-point arithmetic. It is based on the von Neumann's algorithm to sample from the exponential distribution [Neu51], requires no precomputed tables and consumes a smaller amount of random bits than Bernoulli sampling, though it is slower.

Note that none of these methods requires large precomputation depending on the distribution's center c. In all the alternative approaches we present hereafter, there is some large center-dependent precomputation. When the center is not know this can result in prohibitive costs and handling these becomes a major issue around which our work is focused.

2.1.2.2. Center-Dependent Approaches

The cumulative distribution table algorithm is based on the inversion method [Dev86]. All non-negligible cumulative probabilities are stored in a table and at sampling time one generates a cumulative probability in $[0, 1)$ uniformly at random, performs a binary search through the table and returns the corresponding value.

Several alternatives to straightforward CDT are possible. Of special interest are: the alias method [Wal74] which encodes CDTs in a more involved but more efficient approach; BAC Sampling [Saa16] which uses arithmetic coding tables to sample with an optimal consumption of random bits; and Discrete Ziggurat [BCG+14] which adapts from the Ziggurat method [MT84] for a flexible time-memory trade-off.

Finally, Knuth-Yao sampling [KY76] uses a random bit generator to traverse a binary tree formed from the bit representation of the probability of each possible sample, the terminal node is labeled by the corresponding sample. The main advantage of this method is that it consumes a near-optimal

amount of random bits. A block variant and other practical improvements are suggested in [DG14]. This method is strongly center-dependent and it appears difficult to reduce its required precision using the Rényi divergence.

2.1.2.3. Convoluted Sampling

Recently a new sampler was presented in [MW17] using a convolution theorem to combine samples. This algorithm allows to sample according to Gaussian distributions with arbitrary center as follows. Assume the center c has k binary fractionnal digits, i.e. $c \in \mathbb{Z}/2^k$. Then, we can use a first integer Gaussian sampler (scaled by 2^{-k}) to randomly round c to a center in $\mathbb{Z}/2^{k-1}$. Then, we use a second sample (scaled by $2^{-(k-1)}$) to round the new center to a coarser set $\mathbb{Z}/2^{k-2}$, and so on for k times, until we obtain a sample in $\mathbb{Z}$ as desired. Since the final output is obtained by combining a number of Gaussian samples together, the result still follows a Gaussian distribution. Moreover, since the scaling factors grow geometrically, the standard deviation of the final output is (up to a small constant factor) the same as the one of the original samples.

This sampler is currently the most effective in time and memory for large standard deviation. However, it is important to note that the standard deviation for the base sampler has to be greater than the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ [MR07] for the convolution to yield the correct output distribution. Therefore, this approach is not suited to settings where the standard deviation is small, i.e. near to the smoothing parameter as is often the case in lattice-based hash-and-sign signature.

2.2. Twin-CDT, An Arbitrary Centered CDT-based Sampler

In this section we develop techniques from the article *Sampling from Arbitrary Centered Discrete Gaussians for Latiice-Based Cryptography* coauthored with Carlos Aguilar-Melchor and Martin R. Albrecht. We speed-up discrete Gaussian sampling when the center is not known in advance, obtaining a flexible time-memory trade-off comparing favorably to rejection sampling methods. We start with the cumulative distribution table (CDT) suggested in [Pei10] and lower the computational cost of the precomputation phase and the global memory required when sampling from a variable-center discrete Gaussian by precomputing the CDT for a relatively small number of centers ($\mathcal{O}(\lambda^3)$ to be as fast as the fixed-center CDT algorithm) and by computing the cdf when needed, i.e. when for a given uniform random input, the values returned by the CDTs for the two closest precomputed centers differ. Second, we present an adaptation of the lazy technique described in [DN12a] to compute most of the cdf in double IEEE standard double precision, thus decreasing the number of precomputed CDTs. Finally, we propose a more flexible approach which takes advantage of the information already present in the precomputed CDTs. For this we use a Taylor expansion around the precomputed centers and values instead of this lazy technique, thus enabling to reduce the number of precomputed CDTs to a $\omega(\lambda)$ to be as fast as the fixed-center CDT algorithm. We stress, though, that our construction is not constant time, which limits its utility. We address this important issue in section 2.4.1.2.

2.2.1. Variable-Center with Polynomial Number of CDTs

We consider the case in which the mean is variable, i.e. the center is not known before the online phase, as it is the case for lattice-based hash-and-sign signatures. The center can be any real number, but without loss of generality we will only consider centers in $[0, 1)$. Because CDTs are center-dependent, a first naive option would be to precompute a CDT for each possible real

center in $[0, 1)$ in accordance with the desired accuracy. Obviously, this first option has the same time complexity than the classical CDT algorithm, i.e. $\mathcal{O}(\lambda \log s\lambda)$ for λ the security parameter. However, it is completely impractical with 2^λ precomputed CDTs of size $\mathcal{O}(s\lambda^{1.5})$. An opposite trade-off is to compute the CDT on-the-fly, avoiding any precomputation storage, which increase the computational cost to $\mathcal{O}(s\lambda^{3.5})$ assuming that the computation of the exponential function run in $\mathcal{O}(\lambda^3)$ (see Section 2.2.1.2 for a justification of this assumption).

An interesting question is can we keep the time complexity of the classical CDT algorithm with a polynomial number of precomputed CDTs. To answer this question, we start by fixing the number n of equally spaced centers in $[0, 1)$ and precompute the CDTs for each of these. Then, we apply the CDT algorithm to the two precomputed centers closest to the desired center for the same cumulative probability uniformly draw. Assuming that the number of precomputed CDTs is sufficient, the values returned from both CDTs will be equal most of the time, in this case we can conclude, thanks to a simple monotonic argument, that the returned value would have been the same for the CDT at the desired center and return it as a valid sample. Otherwise, the largest value will immediately follow the smallest and we will then have to compute the cdf at the smallest value for the desired center in order to know if the cumulative probability is lower or higher than this cdf. If it is lower then the smaller value will be returned as sample, else it will be the largest.

2.2.1.1. The Twin-CDT Algorithm

As discussed above, to decrease the memory required by the CDT algorithm when the distribution center is determined during the online phase, we can precompute CDTs for a number n of centers equally spaced in $[0, 1)$ and compute the cdf when necessary. The algorithm 2.7, resp. 2.8, describes the offline, resp. online phase, of the *Twin-CDT* algorithm. The algorithm 2.7 precomputes CDTs, up

Algorithm 2.7 Twin-CDT: Offline Phase

Input: a Gaussian parameter s and a number of centers n
Output: a precomputed matrix $\mathbf{T}$

1: initialize an empty matrix $\mathbf{T} \in \mathbb{FP}_\lambda^{n \times 2\lceil ts \rceil + 3}$
2: **for** $i \leftarrow 0, \ldots, n-1$ **do**
3: **for** $j \leftarrow 0, \ldots, 2\lceil ts \rceil + 2$ **do**
4: $\mathbf{T}_{i,j} \leftarrow \mathbb{FP}_m : \mathrm{cdf}_{s,i/n}(j - \lceil ts \rceil - 1)$

to a precision m that guarantees the λ most significant bits of each cdf, and store them with λ bits of precision as a matrix $\mathbf{T}$, where the i-th line is the CDT corresponding to the i-th precomputed center i/n. To sample from $D_{s,c}$, algorithm 2.8 searches the preimages by the cdf of a cumulative probability p, draw from the uniform distribution on $[0, 1)$, in both CDTs corresponding to the center $\lfloor n(c - \lfloor c \rfloor) \rfloor / n$ (respectively $\lceil n(c - \lfloor c \rfloor) \rceil / n$) which return a value v_1 (resp. v_2). If the same value is returned from the both CDTs (i.e. $v_1 = v_2$), then this value added the desired center integer part is a valid sample, else it computes $\mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ and returns $v_1 + \lfloor c \rfloor$ if $p < \mathrm{cdf}_{s,c}(v_1)$ and $v_2 + \lfloor c \rfloor$ else.

Correctness. We establish correctness in the lemma below.

Lemma 2.6. *Assuming that m is large enough to ensure λ correct bits during the cdf computation, the statistical distance between the output distribution of algorithm 2.8 instantiated to sample from $D_{\mathbb{Z}^m,\sigma,c}$ and $D_{\mathbb{Z}^m,\sigma,c}$ is bounded by $2^{-\lambda}$.*

Algorithm 2.8 Twin-CDT: Online Phase

Input: a center c and a precomputed matrix $\mathbf{T}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow 0.U_{\{0,1\}^\lambda}$
$v_1 \leftarrow i - \lceil ts \rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i}$
$v_2 \leftarrow j - \lceil ts \rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $p < \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        return $v_2 + \lfloor c \rfloor$

Proof. First note that from the discrete nature of the considered distribution we have $D_{s,c} = D_{s,c-\lfloor c \rfloor} + \lfloor c \rfloor$. Now recall that the probability integral transform states that if X is a continuous random variable with cumulative distribution function cdf, then $\mathrm{cdf}(X)$ has a uniform distribution on $[0, 1)$. Hence the inversion method: $\mathrm{cdf}^{-1}(U_{[0,1)})$ has the same distribution as X. Finally by noting that for all $s, p \in \mathbb{R}$, $\mathrm{cdf}_{s,c}(p)$ is monotonic in c, if $\mathrm{cdf}^{-1}_{s,c_1}(p) = \mathrm{cdf}^{-1}_{s,c_2}(p) := v$, then $\mathrm{cdf}^{-1}_{s,c}(p) = v$ for all $c \in [c_1, c_2]$, and as a consequence, for all $v \in [-\lceil ts \rceil - 1, \lceil ts \rceil + 1]$, the probability of outputting v is equal to $\mathbb{FP}_m : \mathrm{cdf}_{s,c}(v) - \mathbb{FP}_m : \mathrm{cdf}_{s,c}(v-1)$ which is $2^{-\lambda}$-close to $D_{s,c}(v)$. □

The remaining issue in the correctness analysis of algorithm 2.8 is to determine the error occurring during the m-precision cdf computation. Indeed, this error allows us to learn what precision m is needed to correctly compute the λ most significant bits of the cdf. This error is characterized in Lemma 2.7.

Lemma 2.7. *Let $m \in \mathbb{Z}$ be a positive integer and $\varepsilon = 2^{1-m}$. Let $\bar{c}, \bar{s}, \bar{h} \in \mathbb{FP}_m$ be at distance respectively at most δ_c, δ_c and δ_h from $c, s, h \in \mathbb{R}$ and $h = 1/\rho_{s,c}(\mathbb{Z})$. Let $\Delta f(x) := |\mathbb{FP}_m : f(x) - f(x)|$. We also assume that the following inequalities hold: $s \geq 4$, $t \geq 10$, $s\delta_s \leq 0.01$, $\delta_c \leq 0.01$, $s^2\varepsilon \leq 0.01$, $(ts+1)\varepsilon \leq 1/2$. We have the following error bound on $\Delta\mathrm{cdf}_{s,c}(x)$ for any integer x such that $|x| \leq ts + 2$*

$$\Delta\mathrm{cdf}_{s,c}(x) \leq 3.5t^3s^2\varepsilon$$

Proof. We derive the following bounds using [Duc13, Facts 6.12, 6.14, 6.22]:

$$\Delta\mathrm{cdf}_{s,c}(x) \leq \Delta\left[\sum_{i=-\lceil ts \rceil - 1}^{\lceil ts \rceil + 1} \rho_{s,c}(i)\right]\left(\frac{1}{s} + 3.6s\varepsilon\right) + 3.6s\varepsilon$$

$$\Delta\left[\sum_{i=-\lceil ts \rceil - 1}^{\lceil ts \rceil + 1} \rho_{s,c}(i)\right] \leq 3.2t^3s^3\varepsilon$$

□

For the sake of readability the FPA error bound of Lemma 2.7 is fully simplified and is therefore not tight. For practical implementation, one can derive a better bound using an ad-hoc approach such as done in [PS08].

Efficiency. On average, the evaluation of the cdf requires $\lceil ts \rceil + 1.5$ evaluations of the exponential function. For the sake of clarity, we assume that the exponential function is computed using a direct power series evaluation with schoolbook multiplication, so its time complexity is $\mathcal{O}(\lambda^3)$. We refer the reader to [Bre+06] for a discussion of different ways to compute the exponential function in high-precision.

Lemma 2.8 establishes that the time complexity of the twin-CDT algorithm is $\mathcal{O}(\lambda \log s\lambda + \lambda^4/n)$, so with $n = \mathcal{O}(\lambda^3)$ it has asymptotically the same computational cost than the classical CDT algorithm.

Lemma 2.8. *Let P_{cdf} be the probability of computing the* cdf *during the execution of algorithm 2.8, assuming that $ts \geq 10$, we have*

$$P_{cdf} \leq 2.2ts \left(1 - e^{-\frac{1.25t}{sn}} \Delta_{measure}\right)$$

Proof.

$$\mathsf{P}_{\mathrm{cdf}} \leq \max_{c \in [0,1)} \left(\sum_{i=-\lceil ts \rceil - 1}^{\lceil ts \rceil + 1} \left| \mathrm{cdf}_{s,c}(i) - cdf_{s,c+\frac{1}{n}}(i) \right| \right)$$

Assuming that $ts \geq 10$, we have

$$e^{-\frac{1.25t}{sn}} \Delta_{\mathsf{measure}} \mathrm{cdf}_{s,c}(i) \leq \mathrm{cdf}_{s,c+\frac{1}{n}}(i) \leq \mathrm{cdf}_{s,c}(i)$$

Hence the upper bound. □

On the other hand, the precomputation matrix generated by algorithm 2.7 take n times the size of one CDT, hence the space complexity, which is $\mathcal{O}(ns\lambda^{1.5})$. Note that for n sufficiently big to make the cdf computational cost negligible, the memory space required by this algorithm is about 1 gigabyte for the parameters considered in cryptography and thus prohibitively expensive for practical use.

2.2.1.2. The Lazy-CDT Algorithm

A first idea to decrease the number of precomputed CDTs is to avoid costly cdf evaluations by using the same lazy trick as in [DN12a] for rejection sampling. Indeed, a careful analysis of algorithm 2.8 shows most of the time many of the computed cdf bits are not used. This gives us to a new strategy which consists of computing the bits of $\mathrm{cdf}_{s,c}(v_1)$ lazily. When the values corresponding to the generated probability for the two closest centers are different, the *Lazy-CDT* algorithm first only computes the cdf at a precision m' to ensure $k < \lambda$ correct bits (say $m' = 53$). If the comparison is decided with those k bits, it returns the sample. Otherwise, it recomputes the cdf at a precision m to ensure λ correct bits.

Correctness. In addition to the choice of m, discussed in Section 2.2.1.1, to achieve λ bits of precision, the correctness of algorithm 2.9 also requires to know k which is the number of correct bits after the floating-point computation of the cdf with m' bits of mantissa. For this purpose, given m' Lemma 2.7 provides a theoretical lower bound on k.

Algorithm 2.9 Lazy-CDT: Online Phase

Input: a center c and a precomputed matrix $\mathbf{T}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow 0.U_{\{0,1\}^\lambda}$
$v_1 \leftarrow i - \lceil ts \rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i}$
$v_2 \leftarrow j - \lceil ts \rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $\mathbb{FP}_k : p < \mathbb{FP}_{m'} : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        if $\mathbb{FP}_k : p > \mathbb{FP}_{m'} : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
            return $v_2 + \lfloor c \rfloor$
        else
            if $p > \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
                return $v_1 + \lfloor c \rfloor$
            else
                return $v_2 + \lfloor c \rfloor$

Efficiency. As explained in [DN12a] the precision used for floating-point arithmetic has non-negligible impact, because fp-operation become much expensive when the precision goes over the hardware precision. For instance, modern processors typically provide floating-point arithmetic following the double IEEE standard double precision ($m = 53$), but quad-float FPA ($m = 113$) is usually about 10-20 times slower for basic operations, and the overhead is much more for multiprecision FPA. Therefore the maximal hardware precision is a natural choice for m'. However this choice for m' in algorithm 2.9 is a strong constraint for cryptographic applications, where the error occurring during the floating-point cdf computation is usually greater than 10 bits, making the time-memory tradeoff of algorithm 2.9 inflexible. Note that the probability of triggering high precision in algorithm 2.9 given that $v_1 \neq v_2$ is about $2^{q-k}\mathsf{P}_{\mathsf{cdf}}$, where q is the number of common leading bits of $\mathrm{cdf}_{s,\lfloor n(c-\lfloor c \rfloor)\rfloor/n}(v_1)$ and $\mathrm{cdf}_{s,\lceil n(c-\lfloor c \rfloor)\rceil/n}(v_2)$. By using this lazy trick in addition to lookup tables as described in Section 2.4.1.1 with parameters considered in cryptography, we achieve a computational cost lower than the classical centered CDT algorithm with a memory requirement in the order of 1 megabyte.

2.2.2. A More Flexible Time-Memory Tradeoff

In view of limitations of the lazy approach described above, a natural question is if we can find a better solution to approximate the cdf. The major advantage of this lazy trick is that it does not require additional memory. However, in our context the CDTs are precomputed and rather than approximate the cdf from scratch it would be interesting to reuse the information contained in these precomputations. Consider the cdf as a function of the center and note that each precomputed cdf is zero degree term of the Taylor expansion of the cdf around a precomputed center. Hence, we may approximate the cdf by its Taylor expansions by precomputing some higher degree terms.

At a first glance, this seems to increase the memory requirements of the sampling algorithm, but we will show that this approach allows to drastically reduce the number of precomputed to a $\omega(\lambda)$

centers thanks to a probability which decreases rapidly with the degree of the Taylor expansion. Moreover, this approximation is faster than the cdf lazy computation and it has no strong constraints related to the maximal hardware precision. As a result, we obtain a flexible time-memory tradeoff which reaches, in particular, the same time complexity as the CDT algorithm for centered discrete Gaussians with a practical memory requirements for cryptographic parameters.

We recall the well known Taylor's theorem which provides a polynomial approximation around a given point for any function sufficiently differentiable.

Theorem 2.9 (Taylor's theorem). *Let $d \in \mathbb{Z}^+$ and let the function $f : \mathbb{R} \to \mathbb{R}$ be d times differentiable in some neighborhood U of $a \in \mathbb{R}$. Then for any $x \in U$*

$$f(x) = \mathcal{T}_{d,f,a}(x) + \mathcal{R}_{d,f,a}(x)$$

where

$$\mathcal{T}_{d,f,a}(x) = \sum_{i=0}^{d} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

and

$$\mathcal{R}_{d,f,a}(x) = \int_a^x \frac{f^{(d+1)}(t)}{d!}(x-t)^d dt$$

2.2.2.1. The Taylor-CDT Algorithm

Our *Taylor-CDT* algorithm is similar to the *Lazy-CDT* algorithm (Algorithm 2.9) described above, except that the lazy computation of the cdf is replaced by the Taylor expansion of the cdf, viewed as a function of the Gaussian center, around each precomputed centers for all possible values. The zero-degree term of each of these Taylor expansions is present in the corresponding CDT element $\mathbf{T}_{i,j}$ and the d higher-degree terms are stored as an element $\mathbf{E}_{i,j}$ of another matrix $\mathbf{E}$. As for the other approaches, these precomputations shall be performed at a sufficient precision m to ensure λ correct bits.

Algorithm 2.10 Taylor-CDT: Offline Phase

Input: a Gaussian parameter s, a number of centers n, a Taylor expansion degree d
Output: two precomputed matrices $\mathbf{T}$ and $\mathbf{E}$

1: initialize two empty matrices $\mathbf{T} \in \mathbb{FP}_\lambda^{n \times 2\lceil ts \rceil + 3}$ and $\mathbf{E} \in (\mathbb{FP}_\lambda^d)^{n \times 2\lceil ts \rceil + 3}$
2: **for** $i \leftarrow 0, \ldots, n-1$ **do**
3: **for** $j \leftarrow 0, \ldots, 2\lceil ts \rceil + 2$ **do**
4: $\mathbf{T}_{i,j} \leftarrow \mathbb{FP}_m : \mathrm{cdf}_{s,i/n}(j - \lceil ts \rceil - 1)$
5: $\mathbf{E}_{i,j} \leftarrow \mathbb{FP}_m : \mathcal{T}_{d,\mathrm{cdf}_{s,x}(j-\lceil ts \rceil - 1),i/n}(x) - \mathbf{T}_{i,j}$

During the online phase, algorithm 2.11 proceed as follow. Draw p from the uniform distribution over $[0, 1)$ and search p in the CDTs of the two closest precomputed centers to the desired center decimal part. If the two values found are equal, add the desired center integer part to this value and return it as a valid sample. Otherwise, select the closest precomputed center to the desired center decimal part and evaluate, at the desired center decimal part, the Taylor expansion corresponding to this center and the value found in its CDT. If p is smaller or bigger than this evaluation with respect for the error approximation upper bound $\mathsf{E}_{\mathsf{expansion}}$, characterized in Lemma 2.10, add the desired center integer part to the corresponding value and return it as a valid sample. Otherwise, it is necessary to compute the full cdf to decide which value to return.

Efficiency. Algorithm 2.11 performs two binary searches on CDTs in $\mathcal{O}(\lambda \log s\lambda)$, d additions and multiplications on $\mathbb{FP}_m$ in $\mathcal{O}(m^2)$ with probability $\mathsf{P}_{\mathsf{cdf}} \approx 3\lambda/n$ (see Lemma 2.8) and a cdf computation on $\mathbb{FP}_m$ in $\mathcal{O}(s\lambda^{3.5})$ with probability close to $2^{q+1}\mathsf{P}_{\mathsf{cdf}}\mathsf{E}_{\mathsf{expansion}}$, where q is the number of common leading bits of $\mathrm{cdf}_{s,\lfloor n(c-\lfloor c\rfloor)\rfloor/n}(v_1)$ and $\mathrm{cdf}_{s,\lceil n(c-\lfloor c\rfloor)\rceil/n}(v_2)$ and $\mathsf{E}_{\mathsf{expansion}}$ is the Taylor expansion approximation error bound described in Lemma 2.10.

Algorithm 2.11 Taylor-CDT: Online Phase

Input: a center c and two precomputed matrices $\mathbf{T}$ and $\mathbf{E}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow 0.U_{\{0,1\}^\lambda}$
$v_1 \leftarrow i - \lceil ts\rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c\rfloor)\rfloor,i-1} \le p < \mathbf{T}_{\lfloor n(c-\lfloor c\rfloor)\rfloor,i}$
$v_2 \leftarrow j - \lceil ts\rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c\rfloor)\rceil,j-1} \le p < \mathbf{T}_{\lceil n(c-\lfloor c\rfloor)\rceil,j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c\rfloor$
else
    if $|c - \lfloor n(c - \lfloor c\rfloor)\rfloor| < |c - \lceil n(c - \lfloor c\rfloor)\rceil|$ **then**
        $c' \leftarrow \lfloor n(c - \lfloor c\rfloor)\rfloor$
    else
        $c' \leftarrow \lceil n(c - \lfloor c\rfloor)\rceil$
        $i \leftarrow j$
    if $p < \mathbf{T}_{c',i} + \mathbf{E}_{c',i}(c - \lfloor c\rfloor) - \mathsf{E}_{\mathsf{expansion}}$ **then**
        return $v_1 + \lfloor c\rfloor$
    else
        if $p > \mathbf{T}_{c',i} + \mathbf{E}_{c',i}(c - \lfloor c\rfloor) + \mathsf{E}_{\mathsf{expansion}}$ **then**
            return $v_2 + \lfloor c\rfloor$
        else
            if $p > \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c\rfloor}(v_1)$ **then**
                return $v_1 + \lfloor c\rfloor$
            else
                return $v_2 + \lfloor c\rfloor$

Lemma 2.10. *Let $E_{expansion}$ be the maximal Euclidean distance between $\mathrm{cdf}_{s,x}(v)$ and $\mathcal{T}_{d,\mathrm{cdf}_{s,x}(v),c}(x)$, its Taylor expansion around c, for all $v \in [-\lceil ts\rceil - 1, \lceil ts\rceil + 1]$, $c \in [0,1)$ and $x \in [c, c + 1/2n]$, assuming that $t \ge 2.5$, $s \ge 4$, we have*

$$E_{expansion} < \frac{4t^{d+2}}{n^{d+1}s^{\frac{d+1}{2}}}$$

Proof. From Theorem 2.9 we have

$$\mathsf{E}_{\mathsf{expansion}} = \max_{\substack{c\in[0,1)\\ x\in[c,c+1/2n]\\ v\in[-\lceil ts\rceil-1,\lceil ts\rceil+1]}} \left(\sum_{i=-\lceil ts\rceil-1}^{v} \int_c^x \frac{\rho_{s,t}^{(d+1)}(i)}{d!\,\rho_{s,t}(\mathbb{Z})} (c + \frac{1}{2n} - t)^d \, dt \right)$$

By using well-known series-integral comparison we obtain $\rho_{s,t}(\mathbb{Z}) \ge s\sqrt{2\pi} - 1$ and since $\left|\rho_{s,t}^{(d)}(i)\right| <$

$\frac{d(1.3t)^d 2^d}{s^{d/2}}$ for $s \geq 4$ and $t \geq 2.5$, it follows that

$$\mathsf{E}_{\text{expansion}} \leq \frac{(d+1)(1.3)^{d+1} t^{d+2}}{d!\, n^{d+1} s^{\frac{d+1}{2}}}$$

□

A careful analysis of this technique show that with $d = 4$ we achieve the same asymptotic computational cost as the classical CDT algorithm with $n = \omega(\lambda)$, where the hidden factor is less than $1/4$, therefore for this degree the space complexity of algorithm 2.10 and 2.11 is only λ times bigger than for centered sampling, showing that these algorithms can achieve a memory requirement as low as 1 MB. Finally, note that taking care to add the floating-point computation error to the error of approximation, one can compute the Taylor expansion evaluation at the maximal hardware precision to reduce its computational cost.

2.3. CDT-based Gaussian Sampling: From Multi to Double Precision

In this section we present a new approach to reduce the floating-point precision needed in Gaussian sampling for lattice-based cryptography from the article *CDT-based Gaussian Sampling: From Multi to Double Precision* coauthored with Carlos Aguilar-Melchor. The main idea behind this approach is to reduce the relative error of the cumulative distribution table (CDT) [Pei10] algorithm, then employ a tight bound from [Pre17] for the Rényi divergence of distributions that have a bounded relative error, to obtain a *reordered* CDT algorithm using only standard double precision, while keeping the same security levels, for usual lattice-based signature scheme parameters. This makes it simpler to implement but also more efficient, specially in constant-time. From that, we adapt the Twin-CDT algorithm presented in section 2.2, which is a variable-center variant of the CDT algorithm, to be compatible with our reordered CDTs, and we provide an analysis of its efficiency in this context which show that the amount of floating-point operations is drastically decreased.

2.3.1. Security and Floating-Point Precision

A critical practical issue is how the use of floating point arithmetic affects performance and security. Indeed, as explained in [DN12a] the precision used for floating-point arithmetic has non-negligible impact, because floating-point operations become much more expensive when the requested precision goes over the hardware precision. For instance, modern processors typically provide floating-point arithmetic following the IEEE 754 standard double precision ($p = 53$), but quad-float floating point arithmetic ($p = 113$) is usually about 10-20 times slower for basic operations. The overhead increases rapidly for multiprecision floating point arithmetic.

In [PDG14], a half-CDT was reordered to reduce the relative error in a security analysis of the CDT algorithm centered at zero, using the Kullback-Leibler divergence. In this section, after briefly recalling the CDT algorithm, we also use reordering to reduce the relative error in our setting. Then we present a statistical and Kullback-Leibler distance analysis, as it was done in [PDG14], to establish the floating-point mantissa size p needed in our reordered CDT (rCDT for short) algorithm to maintain the security level of cryptographic schemes using this approximate sampler instead of an ideal sampler. The result of these analysis is that p is too large to fit in IEEE 754 standard double precision arithmetic for lattice-based signature applications. We therefore introduce a

Rényi divergence analysis using techniques from [BLL+15]. Going beyond the Kullback-Leibler analysis reveals to be crucial as, taking into account standard assumptions about the amount of times an attacker can use oracles, we are finally able to show that the precision needed for floating-point arithmetic in Gaussian sampling for lattice-based signature is small enough to use IEEE 754 arithmetic.

2.3.1.1. The CDT Algorithm

It is worth recalling that the correctness of the inverse transformation method comes from the integral transform which states that if X is a continuous random variable with cumulative distribution function cdf, then $\mathrm{cdf}(X)$ has a uniform distribution on $[0, 1)$. Hence the inversion method: $\mathrm{cdf}^{-1}(U_{[0,1)})$ has the same distribution as X. The CDT algorithm [Pei10] is based on this method with the specificity that the image by cdf of the significant domain-subset is precomputed and stored as a cumulative distribution table (CDT). By significant domain-subset we mean an interval

Algorithm 2.12 CDT: Offline Phase

Input: a Gaussian parameter s and a centers c
Output: a precomputed CDT $\mathbf{T}$

1: initialize an empty table $\mathbf{T}$
2: **for** $i \leftarrow 0, \ldots, 2\lceil ts \rceil$ **do**
3: $\quad \mathbf{T}_i \leftarrow \mathrm{cdf}_{s,c}(i - \lceil ts \rceil)$

Algorithm 2.13 CDT: Online Phase

Input: a precomputed CDT $\mathbf{T}$
Output: a sample x that follows $D_{s,c}$

1: $u \leftarrow U_{[0,1)}$
2: **return** $i - \lceil ts \rceil$ s.t. $\mathbf{T}_{i-1} \leq u < \mathbf{T}_i$

large enough in accordance with the desired statistical distance. Indeed, assuming that the CDT values are stored with infinite precision, the statistical distance between the CDT algorithm and an ideal sampler for the same distribution is equal to half of the sum of the probabilities that were not precomputed.

2.3.1.2. Smaller Relative Error

Let $\bar{D}_{s,c}$ be the output distribution of algorithm 2.13, let $S := [-\lceil ts \rceil, \lceil ts \rceil] \cap \mathbb{Z}$ be the truncated support and let p be the floating-point precision, i.e. the number of bits in the mantissa, we have:

$$\delta_{\mathsf{RE}}(D_{s,c}, \bar{D}_{s,c}) \leq \frac{\rho_{s,c}(\mathbb{Z}) - \rho_{s,c}(S)}{2} + \max_{x \in S} \frac{\mathrm{cdf}_{s,c}(x)}{D_{s,c}(x)} 2^{-p}$$

Using a CDT in the natural order, as generated by algorithm 2.12, the relative error of $\bar{D}_{s,c}$ is significantly large due to the fact that in the tail $D_{s,c}(x)$ is very small while $\mathrm{cdf}_{s,c}(x) \approx 1$. A straightforward solution to reduce this relative error would be to reorder the support S such that the smallest probability comes first in the CDT. To this end we use the total order relation $\prec$ on the

support S, defined for any two integers x and y as:

$$x \prec y \text{ if and only if } \begin{cases} D_{s,c}(x) < D_{s,c}(y) \\ D_{s,c}(x) = D_{s,c}(y) \text{ and } x < y \end{cases}.$$

We also denote by $\preceq$ the relation which expands $\prec$ by adding that x is in relation with itself, i.e. $x \preceq y$ if and only if, $x \prec y$ or $x = y$. We denote by $S^{\prec} := (x_i)_{0 \le i \le 2\lceil ts \rceil} := (S, \prec)$ the reordered support according to the order relation $\prec$. Assuming that $c \in [0,1)$[1] the permutation to move from S to $S^{\prec}$ is quite simple:

$$\begin{aligned} S^{\prec} &= ((-1)^{i+1}(\lceil ts \rceil - i))_{0 \le i \le 2\lceil ts \rceil - 2} \| (b_0, b_1) \\ &= (-\lceil ts \rceil, \lceil ts \rceil, -\lceil ts \rceil + 1, \lceil ts \rceil - 1, \ldots, -2, 2, -1, b_0, b_1) \end{aligned}$$

where

$$(b_0, b_1) = \begin{cases} (1,0) & \text{if } 0 \le c < \frac{1}{2} \\ (0,1) & \text{if } \frac{1}{2} \le c < 1 \end{cases}$$

Now we can define a new cumulative distribution function of $D_{s,c}$ over $S^{\prec}$, for any integer $x \in S$, as:

$$\mathrm{rcdf}_{s,c}(x) := \sum_{y \in S}^{y \preceq x} D_{s,c}(y).$$

Note that we assume in the rest of this section that the tailcut parameter t is large enough to ensure the term due to the support truncation $(\rho_{s,c}(\mathbb{Z}) - \rho_{s,c}(S))/2$ is smaller than the precision-dependent term in the relative error of any output distribution. We also modify algorithms 2.12 and 2.13 into algorithms 2.14 and 2.15 to use rcdf instead of cdf. The lemma above shows that the reordered

Algorithm 2.14 Reordered CDT: Offline Phase

Input: a Gaussian parameter s, a center c and a reordered truncated support $S^{\prec} = (x_i)_{0 \le i \le 2\lceil ts \rceil}$
Output: a precomputed reordered CDT $\mathbf{T}$

1: initialize an empty table $\mathbf{T}$
2: **for** $i \leftarrow 0, \ldots, 2\lceil ts \rceil$ **do**
3: $\quad \mathbf{T}_i \leftarrow \mathrm{rcdf}_{s,c}(x_i)$

Algorithm 2.15 Reordered CDT: Online Phase

Input: a precomputed reordered CDT $\mathbf{T}$ and a reordered truncated support $S^{\prec} = (x_i)_{0 \le i \le 2\lceil ts \rceil}$
Output: a sample x that follows $D_{s,c}$

1: $u \leftarrow U_{[0,1]}$
2: **return** x_i s.t. $\mathbf{T}_{i-1} \le u < \mathbf{T}_i$

CDT (rCDT) algorithm decreases the relative error of stored cumulative probabilities by a factor up to $2s$.

[1] It is easy to reduce any center $c \in \mathbb{R}$ to $c' := c - |c| \in [0,1)$ without loss of generality for any probability distribution over the integers.

Lemma 2.11. *Let $\bar{D}^{\prec}_{s,c}$ be the output distribution of algorithm 2.15 and p be the number of bits in the mantissa for each floating-point number used in algorithm 2.15, for $s \geq 1$ we have:*

$$\delta_{RE}(D_{s,c}, \bar{D}^{\prec}_{s,c}) < 2^{-p+2.3+\log_2 s}$$

Proof. Let S^-, S^+ be respectively the negative part and the strictly positive part of S, i.e. $S^- := [-\lceil ts \rceil, 0] \cap \mathbb{Z}$ and $S^+ := [1, \lceil ts \rceil] \cap \mathbb{Z}$. We know that:

$$\frac{\mathrm{rcdf}_{s,c}(x)}{D_{s,c}(x)} = \sum_{y \in S^-}^{y \preceq x} \frac{D_{s,c}(y)}{D_{s,c}(x)} + \sum_{z \in S^+}^{z \preceq x} \frac{D_{s,c}(z)}{D_{s,c}(x)}$$

And from a standard sum-integral comparison argument:

$$\begin{aligned} \sum_{y \in S^-}^{y \preceq x} \frac{D_{s,c}(y)}{D_{s,c}(x)} + \sum_{z \in S^+}^{z \preceq x} \frac{D_{s,c}(z)}{D_{s,c}(x)} &\leq 2 + 2 \int_{y=|x|}^{\infty} \frac{\rho_{s,c}(y)}{\rho_{s,c}(|x|)} \\ &\leq 2 + s\sqrt{2\pi} \end{aligned}$$

□

2.3.1.3. Security Analysis

In previous works, the precision was determined by an analysis based on the statistical distance [Pei10] or the Kullback-Leibler divergence [PDG14] for classical CDT and reversed half-CDT. In this section, the distributions Φ and Φ' denote the cryptographic scheme in the view of the adversary in the approximate (resp. ideal) cases. We assume that a query to the private-key functionality oracle corresponds to m queries to the Gaussian sampling algorithm, with a maximum Gaussian parameter $s := \max_{i=1,\ldots,m} s_i$, and, in accordance with the NIST call for proposals[2] for the post-quantum cryptography standardization, we bound the number of queries from the attacker to the private-key functionality oracle by q_s. Note that in the NIST call for proposals $q_s = 2^{64}$.

Statistical distance analysis. Any adversary with success probability ε' on the scheme implemented with perfect Gaussian sampling has a success probability $\varepsilon \leq \varepsilon' + \Delta(\Phi, \Phi')$ against the scheme implemented with approximative Gaussian sampling. We assume that the parameters for the ideal scheme are selected to have $\varepsilon' \leq 2^{-\lambda-1}$. To ensure a security against mq_s queries, each of the approximated Gaussian random variables $(\bar{D}_{s_i,c_i})_i$ should be within statistical distance $\Delta(\Phi, \Phi')/(mq_s)$ of the desired $(D_{s_i,c_i})_i$. Using $\Delta(\bar{D}_{s_i,c_i}, D_{s_i,c_i}) \leq 2^{-p+1.3+\log_2 s_i}$ from Lemma 2.11 leads to a mantissa precision requirement on the rCDT for λ-bits of security:

$$p \geq \lambda + 2.3 + \log_2(smq_s).$$

Kullback-Leibler divergence analysis. In [PDG14] the statistical distance analysis is replaced by the Kullback-Leibler divergence, i.e. the Rényi divergence of order $a = 1$, to reduce the precision p needed in the precomputed table. They show that any adversary with success probability $\varepsilon' \leq 2^{-\lambda-1}$ on the scheme implemented with perfect Gaussian sampling has a success probability

[2] http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf

$\varepsilon \leq \varepsilon' + \sqrt{\log R_1(\Phi\|\Phi')/2}$ against the scheme implemented with approximative Gaussian sampling. We assume that the parameters for the ideal scheme are selected to have $\varepsilon' \leq 2^{-\lambda-1}$. By the multiplicative property of the Rényi divergence over the mq_s independant samples, we have $R_1(\Phi\|\Phi') \leq (\max_{i=1,\ldots,m} R_1(\bar{D}_{s_i,c_i}\|D_{s_i,c_i}))^{mq_s}$. Using $R_1(\bar{D}_{s_i,c_i}\|D_{s_i,c_i}) \leq \exp(2^{-2p+4.6+2\log_2 s})$ from Lemmas 2.11 and 2.2 leads to a mantissa precision requirement on the rCDT for λ-bits of security:

$$p \geq \lambda + 3.3 + \log_2(s\sqrt{mq_s}).$$

Rényi divergence analysis. As described in [BLL+15], the probability preservation property of the Rényi divergence is multiplicative for $a > 1$ rather than additive for $a = 1$ and the statistical distance. Any adversary with success probability ε' on the scheme implemented with perfect Gaussian has a success probability $\varepsilon \leq (\varepsilon' R_a(\Phi\|\Phi'))^{\frac{a-1}{a}}$ against the scheme implemented with approximative Gaussian sampling. We assume that the parameters for the ideal scheme are selected to have $\varepsilon' \leq 2^{-\frac{a}{a-1}\lambda-1}$, i.e. assuming that for any $k > 0$ we have $\lambda \leq (a-1)k$, then the parameters for the ideal scheme are selected to have $(\lambda + k + 1)$-bits of security. By the multiplicative property of the Rényi divergence we get that $R_a(\Phi\|\Phi') \leq (\max_{i=1,\ldots,m} R_a(\bar{D}_{s_i,c_i}\|D_{s_i,c_i}))^{mq_s}$. Using Lemmas 2.11 and 2.3 leads to a mantissa precision requirement on the rCDT for λ-bits of security:

$$p \geq 1.3 + \log_2(s\sqrt{amq_s})$$

For instance if we take $a = 256$, we can take $p = 53$, i.e. the standard IEEE 754 double precision, as long as we have $\log_2(s\sqrt{mq_s}) \leq 47.7$, which is the case for usual lattice-based signature schemes where $q_s = 2^{64}$, $m \leq 2^{11}$ and $s \leq 2^8$.

Comparing the presented security analysis (statistical distance, Kullback-Leibler divergence and Rényi divergence), we conclude that the Rényi divergence analysis results in the smallest mantissa size p. Moreover, this size is, in the Rényi divergence analysis, independent of the targeted security parameter which is a nice side effect. With $p = 53$, i.e. the standard IEEE 754 double precision, this leads to $128\lceil s\sqrt{\lambda}/2\rceil + 64$ bits of memory for a full CDT and $64\lceil s\sqrt{\lambda}/2\rceil + 64$ bits for half a CDT[3]. This results in a significant reduction of the memory needed by (reordered) CDT algorithm with respect to the other approaches. Besides this memory reduction, an important consequence of considering standard IEEE 754 double precision is that this allows to store each CDT entry in only one register which makes the resistance to the timing and cache attacks easier to achieve and limits its overhead.

2.3.2. Double Precision Variable Center CDT Algorithm

Now we consider the case in which the center is not know before the online phase, as it is the case for lattice-based hash-and-sign signatures. Indeed, to sample from a discrete Gaussian distribution $D_{s,c}$, the (reordered) CDT algorithm requires knowledge of the target distribution's center c during the offline phase. However, for lattice-based cryptographic hash-and-sign signatures, the distribution center depends on the message to be signed, i.e. it is determined during the online phase. Note that the center can be any real number, but from the discrete nature of the considered distribution we have $D_{s,c} = D_{s,c-\lfloor c\rfloor} + \lfloor c\rfloor$, therefore we will only consider centers in $[0, 1)$. Because CDTs are

[3]When the considered Gaussian density function is symmetric, half a CDT can be stored instead of the full CDT. The method then proceeds to sample from the half distribution and draws a random bit to determine the sign. Note that this is the setting that has been considered for the CDT presented in [PDG14] with the Kullback-Leibler divergence.

center-dependent, a first naive option, as discussed in section 2.2, would be to precompute a CDT for each possible real center in $[0, 1)$ in accordance with the desired accuracy. Obviously, this first option has the same time complexity than the classical CDT algorithm. However, it is completely impractical with an exponential number of precomputed CDTs. An opposite trade-off is to compute the CDT on-the-fly, avoiding any precomputation storage while sacrificing efficiency with several floating-point computations of the exponential function for each sample. In this section we adapt the Twin-CDT algorithm to our reordered cdf (rcdf), then we analyze its efficiency and propose a trick to implement it in constant time.

2.3.2.1. The Double Precision Twin-CDT Algorithm

Let n be a time-memory trade-off parameter corresponding to the number of equally spaced centers in $[0, 1)$ for which CDTs are precomputed. The reordered Twin-CDT algorithm applies the reordered CDT algorithm (algorithm 2.15) to the two precomputed centers closest to the desired center. Assuming that n is large enough, values returned from both CDTs will be equal most of the time. In this case we can conclude, thanks to a simple monotonic argument, that the returned value would have been the same for the CDT at the desired center. Otherwise, in accordance with the order relation $\prec$ defined in Section 2.3.1.2, the largest returned value will immediately follow the smallest and we will then have to compute the rcdf at the smallest value for the desired center in order to know if the cumulative probability is lower or higher than this rcdf. If it is lower then the smaller value will be returned as sample, else it will be the largest. Algorithm 2.16 and 2.17 describe respectively the offline and online phases of this double precision Twin-CDT algorithm in our context. Algorithm 2.16 precomputes these CDTs and store them as a matrix $\mathbf{T}$, where the i-th line is the CDT corresponding to the i-th precomputed center i/n. To sample from $D_{s,c}$, algorithm 2.17

Algorithm 2.16 Double Precision Twin-CDT: Offline Phase

Input: a Gaussian parameter s and a number of centers n and a reordered truncated support $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Output: a precomputed matrix $\mathbf{T}$

```
initialize an empty matrix T
for i ← 0, ..., n − 1 do
    for j ← 0, ..., 2⌈ts⌉ do
        T_{i,j} ← rcdf_{s,i/n}(x_j)
```

searches the preimages by the cdf of a cumulative probability u, draw from the uniform distribution on $[0, 1)$, in both CDTs corresponding to the center $\lfloor n(c - \lfloor c \rfloor) \rfloor / n$ (respectively $\lceil n(c - \lfloor c \rfloor) \rceil / n$) which return a value v_1 (resp. v_2). If the same value is returned from the both CDTs (i.e. $v_1 = v_2$), then this value, added to the desired center integer part, is a valid sample, else one computes $\mathrm{rcdf}_{s,c-\lfloor c \rfloor}(v_1)$ and returns $v_1 + \lfloor c \rfloor$ if $u < \mathrm{rcdf}_{s,c}(v_1)$ and $v_2 + \lfloor c \rfloor$ else[4].

Correctness. The precision needed in the double precision Twin-CDT algorithm (algorithms 2.16 and 2.17) to achieve a given level of security is the same as for the reordered-CDT algorithm (algorithms 2.14 and 2.15) as discussed in Section 2.3.1.3. We establish correctness of algorithm 2.17 in the lemma below.

[4]When n is too small to have $|v_1 - v_2| \leq 1$ for any $u \in [0, 1]$, we have to compute at most $|v_1 - v_2|$ evaluations of the rcdf function to conclude.

Algorithm 2.17 Double Precision Twin-CDT: Online Phase

Input: a center c, a precomputed matrix $\mathbf{T}$ and a reordered truncated support $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Output: a sample x that follows $D_{s,c}$

$u \leftarrow U_{[0,1)}$
$v_1 \leftarrow x_i$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i-1} \leq u < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i}$
$v_2 \leftarrow x_j$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j-1} \leq u < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $u < \mathrm{rcdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        return $v_2 + \lfloor c \rfloor$

Lemma 2.12. *Assuming that the calculations are done with infinite precision. If the precomputed matrix* $\mathbf{T}$ *has been generated by algorithm 2.16 instantiated with* $\mathrm{rcdf}_{s,z}(x) := \sum_{y \in S}^{y \prec x} D_{s,z}(y)$*, then the output distribution of algorithm 2.17 is* $D_{s,c}$ *for any given input* $c \in \mathbb{R}$*.*

Proof. By noting that for all $s, p \in \mathbb{R}$, $\mathrm{rcdf}_{s,c}^{-1}(p)$ is monotonic in c, we have: if $\mathrm{rcdf}_{s,c_1}^{-1}(p) = \mathrm{rcdf}_{s,c_2}^{-1}(p) := x_i$, then $\mathrm{rcdf}_{s,c}^{-1}(p) = x_i$ for all $c \in [c_1, c_2]$ and, as a consequence, for all $x_i \in S^{\prec}$, the probability of outputting x_i is equal to $\mathrm{rcdf}_{s,c}(x_i) - \mathrm{rcdf}_{s,c}(x_{i-1})$ which is equal to $D_{s,c}(x_i)$. □

Efficiency. An important issue in the efficiency analysis of Algortihm 2.17 is how small is the probability of computing the rcdf function. An upper bound is presented in section 2.2 for classical CDTs. In the lemma below we show that this probability is less than $6\sqrt{\lambda}/n$ for reordered CDTs, even for small values of s (say $s \approx 1$).

Lemma 2.13. *Let* P_{rcdf} *be the probability of computing the* rcdf *during the execution of algorithm 2.17 instantiated to sample from* $D_{s,c}$*. Assuming* $s \geq 1$, $t \geq 4$, $n \geq \frac{t}{s} + \frac{3}{s^2}$, $\epsilon \leq \frac{1}{40nt}$ *and* $E_{tailcut} \leq \frac{1}{20nst}$*, we have*

$$P_{\mathrm{rcdf}} \leq \frac{12t}{ns}$$

Proof. We first note that Algortihm 2.17 compute rcdf only when $\mathrm{rcdf}_{s,c}^{-1}(u) \neq \mathrm{rcdf}_{s,c+\frac{1}{n}}^{-1}(u)$. Hence the following bound:

$$\begin{aligned}
\mathsf{P}_{\mathrm{rcdf}} &\leq \sum_{x \in S} \left| \mathrm{rcdf}_{s,c}(x) - \mathrm{rcdf}_{s,c+\frac{1}{n}}(x) \right| \\
&\leq \sum_{x \in S} \sum_{y \in S}^{y \preceq x} \left| \frac{\rho_{s,c}(y)}{\rho_{s,c}(S)} - \frac{\rho_{s,c+\frac{1}{n}}(y)}{\rho_{s,c+\frac{1}{n}}(S)} \right| \\
&\leq \sum_{x_i \in S^{\prec}} (2\lceil ts \rceil - i + 1) \left| \frac{\rho_{s,c}(x_i)}{\rho_{s,c}(S)} - \frac{\rho_{s,c+\frac{1}{n}}(x_i)}{\rho_{s,c+\frac{1}{n}}(S)} \right|
\end{aligned}$$

Then, we use Lemma 2.5 and Lemma 2.4 to deal with the distances between Gaussian measures

$\rho_{s,c}(S)$, $\rho_{s,c}(\mathbb{Z})$, $\rho_{s,c+\frac{1}{n}}(\mathbb{Z})$ and $\rho_{s,c+\frac{1}{n}}(S)$:

$$\left|\frac{\rho_{s,c}(x_i)}{\rho_{s,c}(S)} - \frac{\rho_{s,c+\frac{1}{n}}(x_i)}{\rho_{s,c+\frac{1}{n}}(S)}\right| \leq \left|\frac{\rho_{s,c}(x_i)}{\rho_{s,c}(\mathbb{Z})} - \frac{\rho_{s,c+\frac{1}{n}}(x_i)}{\rho_{s,c+\frac{1}{n}}(\mathbb{Z})}\right| + 2\mathsf{E}_{\mathsf{tailcut}}$$

and

$$\left|\frac{\rho_{s,c}(x_i)}{\rho_{s,c}(\mathbb{Z})} - \frac{\rho_{s,c+\frac{1}{n}}(x_i)}{\rho_{s,c+\frac{1}{n}}(\mathbb{Z})}\right| \leq \left|\frac{\rho_{s,c}(x_i) - \rho_{s,c+\frac{1}{n}}(x_i) + 4\epsilon}{\rho_{s,c}(\mathbb{Z})}\right|.$$

Assuming that $n \geq \frac{t}{s} + \frac{3}{s^2}$ and using that $\rho_{s,c+\frac{1}{n}}(x) = \rho_{s,c}(x)\exp\left(\frac{x_i-c}{2ns^2} - \frac{1}{2n^2s^2}\right)$ leads to:

$$\rho_{s,c}(x_i) - \rho_{s,c+\frac{1}{n}}(x_i) \leq \left(\frac{x_i - c}{ns^2} - \frac{1}{n^2s^2}\right)\rho_{s,c}(x_i)$$

Finally, we have:

$$\sum_{x_i \in S^{\prec}} (2\lceil ts\rceil - i + 1)\,\rho_{s,c}(x_i) \leq 3ts$$

and from $\rho_{s,c}(x) \leq e^{-k}$ for all $x \leq \sqrt{2}ks$:

$$\begin{aligned}\sum_{x\in S}\sum_{y\in S}^{y\preceq x} y\rho_{s,c}(y) &\leq \sum_{\substack{x\in S\\ x\leq 1}} x\rho_{s,c}(x)\\ &\leq \frac{1}{\sqrt{e}} + \sum_{k=0}^{t}\left(e^{-k}\sum_{x=\lceil\sqrt{2}ks\rceil}^{\lfloor\sqrt{2}(k-1)s\rfloor} x\right)\\ &\leq 8.1s^2\end{aligned}$$

Hence the upper bound. □

From Lemma 2.13 we have the average time complexity of the reordered Twin-CDT sampler (algorithm 2.17) which, assuming that the double precision is enough according to Section 2.3.1.3, is $O(\frac{\lambda}{n}\mathcal{C}_{\exp} + \log s\sqrt{\lambda})$, where $\mathcal{C}_{\exp}$ is the time complexity of computing the exponential function. Note that the $O(\log s\sqrt{\lambda})$ term is due to the search in the two CDTs using a binary search algorithm, there is a factor $\sqrt{\lambda}/n$ in front of $\mathcal{C}_{\exp}$ for the P_{rcdf} and another factor $s\sqrt{\lambda}$ for the $O(st)$ evaluations of $\exp$ needed to compute the rcdf. About its space complexity, the precomputation matrix generated by algorithm 2.16 takes clearly n times the size of one CDT, hence a space complexity of $O(ns\sqrt{\lambda})$.

2.3.2.2. CDF Approximation

The main drawback in the (double precision) Twin-CDT algorithm describe above is the costly computation of the (reordered) cdf which needs about ts evaluations of the exponential function in average. The idea to avoid cdf evaluations is to use an approximation of the cdf whose evaluation is faster than that of cdf. At the first glance, this seems to increase the memory needed by the sampling algorithm, but as shown in section 2.2 the running time saving may allow to reduce the number of precomputed centers thanks to the fast convergence.

In section 2.2 we propose to evaluate a precomputed sum of Taylor expansions instead of the cdf in the Twin-CDT algorithm. This approach allows a more flexible time-memory trade-off which takes advantage of the information already present in the precomputed CDTs. In the same way, by considering the cdf as a function of the center, we note that each precomputed $\mathrm{rcdf}_{s,c}(x)$ is the Taylor expansion zero degree term of the cdf around the value x and the precomputed center c. Let $\mathcal{T}_{d,\mathrm{rcdf}_{s,x}(v),c}(x)$ be the Taylor expansions of the rcdf, viewed as a function of the Gaussian center x, around each precomputed center c for all possible values v. The zero-degree term of each of these Taylor expansions is present in the corresponding rCDT element $\mathbf{T}_{i,j}$ and the d higher-degree terms are stored as an element $\mathbf{E}_{i,j}$ of another matrix $\mathbf{E}$. During the online phase, algorithm 2.19 proceeds

Algorithm 2.18 Double Precision Taylor-CDT: Offline Phase

Input: a Gaussian parameter s, a number of centers n, a Taylor expansion degree d and a reordered truncated support $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Output: two precomputed matrices $\mathbf{T}$ and $\mathbf{E}$

initialize two empty matrices $\mathbf{T}$ and $\mathbf{E}$
for $i \leftarrow 0, \ldots, n-1$ **do**
    for $j \leftarrow 0, \ldots, 2\lceil \tau s \rceil$ **do**
        $\mathbf{T}_{i,j} \leftarrow \mathrm{rcdf}_{s,i/n}(x_j)$
        $\mathbf{E}_{i,j} \leftarrow \mathcal{T}_{d,\mathrm{rcdf}_{s,x}(x_j),i/n}(x) - \mathbf{T}_{i,j}$

as follows. Draw u from the uniform distribution over $[0,1)$ and search u in the rCDTs of the two closest precomputed centers to the desired center decimal part. If the two values found are equal ($v_1 = v_2$), add the desired center integer part to this value and return it as a valid sample. Otherwise ($v_1 < v_2$), evaluate, at the desired center decimal part, the Taylor expansion corresponding to v_1 and the center having returned it. If u is smaller than this evaluation add the desired center integer part to v_1 and return it as a valid sample. Otherwise, add the desired center integer part to v_2 and return it.

Algorithm 2.19 Double Precision Taylor-CDT: Online Phase

Input: a center c, two precomputed matrices $\mathbf{T}, \mathbf{E}$ and a reordered truncated support $S^{\prec} = (x_i)_{0 \leq i \leq 2\lceil ts \rceil}$
Output: a sample x that follows $D_{s,c}$

$u \leftarrow U_{[0,1)}$
$v_1 \leftarrow x_i$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i}$
$v_2 \leftarrow x_j$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor) \rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor) \rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $u < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i} + \mathbf{E}_{\lfloor n(c-\lfloor c \rfloor) \rfloor, i}(c - \lfloor c \rfloor)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        return $v_2 + \lfloor c \rfloor$

Correctness. From Lemma 2.10, assuming $d \geq 2$, algorithm 2.19 is correct for:

$$d \geq \frac{54}{\log_2 n + \frac{1}{2}\log_2 s - \log_2 \lambda}$$

Note that another option consists in evaluating the Taylor expansion with a smaller degree d' and returning the sampler only if the approximation error allows it and otherwise evaluate the rcdf.

This Taylor approximation allows us to evaluate only a small degree polynomial instead of the rcdf which, since the time complexity of the rcdf evaluation depends on s, is particularly interesting when s is large. Moreover, the approximation error decreases rapidly with d, which helps to decrease the number of precomputed CDTs n and, as a consequence, the memory required for the execution of the (double precision) Twin-CDT sampler. This remains true even if we take into account the additional precomputation due to the d coefficients of the Taylor expansion.

2.4. Implementation and Performances

In this section we present implementation techniques and the performances of our implementations.

2.4.1. Implementation

In this section we present a lookup-table trick and two techniques to protect implementations against *timing-attacks* based on statistical arguments, we bound the amount of worst-case situations that can occur. As a result, we conclude that constant-time and cache-resistant variants are also viable.

2.4.1.1. Lookup Tables

We shall now show how to use partial lookup tables to avoid the binary search in most cases when using CDT algorithms, this technique is the CDT analogue of the Knuth-Yao algorithm improvement described in [CRVV15]. Note that this strategy is particularly fitting for discrete Gaussian distributions with relatively small expected values. The basic idea is to subdivide the uniform distribution $U_{[0,1]}$ into ℓ uniform distributions on subsets of the same size $U_{[i/\ell,(i+1)/\ell]}$, with ℓ a power of two. We then precompute a partial lookup table on these subsets which allows to return the sample at once when the subset considered does not include a cdf image. We note that instead of subdividing the uniform range into stripes of the same size, we can also recursively subdivide only some stripes of the previous subdivision. However, for the sake of clarity and ease of exposure, this improvement is not included in this paper and we will describe this technique for the classical centered CDT algorithm.

First, we initialize a lookup table of size $\ell = 2^l$ where the i-th entry corresponds to a subinterval $[i/\ell, (i+1)/\ell]$ of $[0, 1]$. Second, after precomputing the CDT, we mark all the entries for which there is at least one CDT element in their corresponding subinterval $[i/\ell, (i+1)/\ell]$ with $\perp$, and all remaining entries with $\top$. Each entry marked with $\top$ allows to return a sample without the need to perform a binary search in the CDT, because only one value corresponds to this subinterval which is the first CDT element greater or equal to $(i+1)/\ell$.

Efficiency. The efficiency of this technique is directly related to the number of entries, marked with $\top$, whose subintervals do not contain a CDT element. We denote the probability of performing binary search by $\mathsf{P}_{\mathsf{binsrch}}$, obviously the probability to return the sample immediately after choosing i, which is a part of p, is $1 - \mathsf{P}_{\mathsf{binsrch}}$. Lemma 2.14 gives a lower bound of $\mathsf{P}_{\mathsf{binsrch}}$.

Lemma 2.14. *For any $\ell \geq 2^8$ and $s \geq \eta_{\frac{1}{2}}(\mathbb{Z})$. Let $P_{binsrch}$ be the probability of performing binary search during the execution of the CDT algorithm implemented with the lookup table trick described above, we have*

$$P_{binsrch} < 1.2s\sqrt{\log_2 \ell}/\ell$$

Proof.

$$\mathsf{P}_{\mathsf{binsrch}} = \frac{\ell - \sum_{i=\lfloor c-\tau s \rfloor}^{\lceil c+\tau s \rceil} \lfloor \ell \, \mathrm{cdf}_{s,c}(i) \rfloor - \lfloor \ell \, \mathrm{cdf}_{s,c}(i-1) \rfloor}{\ell}$$

From Lemma 2.5 we have

$$\left\lfloor \ell \, \mathrm{cdf}_{s,c}\left(\left\lfloor c - 0.6s\sqrt{\log_2 \ell} \right\rfloor\right) \right\rfloor = 0$$

$$\left\lfloor \ell \left(1 - \mathrm{cdf}_{s,c}\left(\left\lceil c + 0.6s\sqrt{\log_2 \ell} \right\rceil\right)\right) \right\rfloor = 0$$

□

2.4.1.2. Constant Time

A general problem with sampling algorithms in cryptography is that the running time can leak information about the output sample or the input, which clearly hurts security. A first potential source of leakage is the search algorithm used in the (Twin) CDT algorithm, indeed the usual binary search algorithm stops when the target value matches the middle element, which can leak information about the output sample. A simple workaround is to continue the search in the lower or upper half of the array eliminating the other half from consideration, as if the target value had not been found. Note that this modification does not change the time complexity of the binary search algorithm, i.e. $\mathcal{O}(\log(ts))$ in our context.

To have a constant-time implementation of the (double precision) Twin-CDT, we have to carefully use a constant-time evaluation or approximation of the rcdf, and we have to handle the worst-case computations of this algorithm, in the sense that this part is executed only with probability q (depending if $v_1 \neq v_2$ for the Twin-CDT algorithm). Note that the worst-case computational cost of rcdf can be very large but our algorithm exploits the fact that the probability of this worst-case is low enough so that the average cost is reasonable.

In order to hide when these worst-case computations are done a naive solution would be to do them on every iteration of the protocol. This would result in a complete loss of the advantages brought by our approach and raise the computational cost to the one of a naive on-the-fly CDT approach. The problem is that the probability of the worst-case situation is low but not *cryptographically*-low and thus for a single iteration of the sampling algorithm the naive solution is the only solution.

Fortunately, in cryptographic applications of Gaussian sampling, we need vectors (with thousands of coordinates) of samples and thus we can use probabilistic tools to limit the amount of worst-case situations that can occur. The probability that over many samples, the amount of worst-case situations deviates by a multiplicative factor from the expected value can be shown to be cryptographically small even for small factors. We thus can pad the amount of such worst-case situations without increasing their impact on the average cost significantly.

More explicitly, we take advantage of the fact that the cryptographic scheme samples vectors of m variables at once to pad this worst-case part over the m executions of the sampling algorithm. Let K be the number of times the worst-case part has been executed during m executions of the algorithm and q, as already stated, the probability that the worst-case part needs to be run on one iteration. From Hoeffding's inequality we have:

$$\Pr_{K\sim\mathcal{B}(m,q)}[K > m(x+q)] \leq 2e^{-2mx^2}.$$

Thus, if we pad the amount of worst-case computations to $m(x+q)$ with $2mx^2 = \lambda$ the probability that the padding will not suffice and will result in a non-constant time execution will be bounded by $2^{-\lambda}$ which is clearly enough for cryptographic applications. The expected amount of worst-case computations is mq and thus the computational overhead for this padded m sample algorithm is of mx worst-case computations. Thus the per sample overhead is of $x = \sqrt{\lambda/2m}$ worst-case computations. As typically $m \gg \lambda$ the overhead in this approach is in practice not significant.

2.4.1.3. Cache attacks

With a worst-case padding technique, the amount of worst-case operations done is constant (except with cryptographically low probability in our case). However, a cache attack technique, such as used in [BHLY16] could be used to infer which were the samples for which a worst-case operation would really have been done. The idea is that if we run m sampling algorithms refusing to do the worst-case operations, and only after do all the worst-case operations in a row and write back the results to the samples that really required the worst-case computation, pattern accesses (and thus cache attacks) will completely break the usefulness of the padding we did.

In practice, as shown in the experimental results in the section 2.4.2, the per sample computational cost is on dozens of cycles for cryptographical applications. In this case, it is easy to get protected against cache attacks. For example, the worst-case inputs (resp. results) can be read from (resp. written back to) the m samples using a Square-root ORAM protocol [ZWR+16] which will cost in the average $\sqrt{m}$ memory accesses per read/write operation with total pattern access protection. Again as m is typically in the thousands, $\sqrt{m}$ memory access will not change significantly the amount of cycles needed per sample. Considering cache attacks on the regular-case computations, the sizes being of same order, overhead would be of same magnitude.

2.4.2. Performances

In this section, we present the performances of our open-source (GPLv3+) C++ implementation[5] of the multi-precision twin-CDT sampler which uses the MPFR [FHL+07] and GMP [Gt15] libraries as well as Salsa20 [Ber08] as the pseudorandom number generator. We also present the performances of our C implementation of the double-precision twin-CDT sampler which use Salsa20 [Ber08] as the pseudorandom number generator. The table 2.1 present a comparison with traightforward and Karney rejection methods implemented in the dgs library [Alb14].

2.4.2.1. Multi-Precision Twin-CDT

Our non-centered discrete Gaussian sampler was implemented with a binary search executed byte by byte if $\ell = 2^8$ and 2-bytes by 2-bytes if $\ell = 2^{16}$ without recursive subdivision of $U_{[0,1]}$, therefore $[0, 1]$

[5]The implementation is available at `https://github.com/tricosset/FGN`.

Table 2.1. – Comparison of the Twin-CDT algorithm (for $n \in \{3, 10, 100\}$ precomputed centers and a security parameter $\lambda = 256$ which determines the tailcut parameter $t = \sqrt{\lambda}/2$) with straightforward and Karney rejection methods (both from the dgs library [Alb14]). All timings are on a 2.90GHz Intel(R) Core(R) i5–4210H, use one core and do not include the pseudorandom generation.

Gaussian width parameter	Algorithm	Samples per second	
		Multi-precision	Double-precision
2	Twin-CDT-100	2 581 195	52 935 721
	Twin-CDT-10	1 443 257	43 104 933
	Twin-CDT-3	673 397	35 086 446
	Karney	40 359	775 795
	Rejection	10 389	122 564
4	Twin-CDT-100	2 340 994	46 910 085
	Twin-CDT-10	987 421	38 566 583
	Twin-CDT-3	395 089	31 660 993
	Karney	30 946	623 441
	Rejection	9 451	120 772
8	Twin-CDT-100	2 049 566	42 683 912
	Twin-CDT-10	609 408	36 123 197
	Twin-CDT-3	210 078	25 932 004
	Karney	27 388	587 544
	Rejection	9 308	118 203
16	Twin-CDT-100	1 563 816	36 210 391
	Twin-CDT-10	340 939	25 957 210
	Twin-CDT-3	112 685	21 142 623
	Karney	24 879	557 413
	Rejection	9 685	119 166
32	Twin-CDT-100	1 057 773	29 434 654
	Twin-CDT-10	180 366	27 238 625
	Twin-CDT-3	58 336	24 402 531
	Karney	25 553	567 214
	Rejection	9 492	120 039

is subdivided in ℓ intervals of the same size and $\mathrm{cdf}(x)$ is stored for all $x \in [-\lceil \tau\sigma \rceil - 1, \lceil \tau\sigma \rceil + 1]$. The implementation of our non-centered discrete Gaussian sampler uses a fixed number of precomputed centers $n = 2^8$ with a lookup table of size $\ell = 2^8$ and includes the lazy cdf evaluation optimization. We tested the performance of our non-centered discrete Gaussian sampler by using it

Table 2.2. – Performance of sampling from $D_{(g),\sigma'}$ as implemented in [ACLL15] and with our non-centered discrete Gaussian sampler with $\ell = n = 2^8$. The column $D_{(g),\sigma'}/s$ gives the number of samples returned per second, the column "memory" the maximum amount of memory consumed by the process. All timings are on a Intel(R) Xeon(R) CPU E5-2667 (strombenzin). Precomputation uses 2 cores, the online phase uses one core.

			[ACLL15]		
N	$\log \sigma'$	precomp	time	$D_{(g),\sigma'}/s$	memory
256	38.2	0.08 s	8.46 ms	118.17	11,556 kB
512	42.0	0.17 s	16.96 ms	58.95	11,340 kB
1024	45.8	0.32 s	38.05 ms	26.28	21,424 kB
2048	49.6	0.93 s	78.17 ms	12.79	41,960 kB
4096	53.3	2.26 s	157.53 ms	6.35	86,640 kB
8192	57.0	6.08 s	337.32 ms	2.96	192,520 kB
16384	60.7	13.36 s	700.75 ms	1.43	301,200 kB
			this work		
N	$\log \sigma'$	precomp	time	$D_{(g),\sigma'}/s$	memory
256	38.2	0.31 s	2.91 ms	343.16	17,080 kB
512	42.0	0.39 s	5.99 ms	166.88	21,276 kB
1024	45.8	0.65 s	11.89 ms	84.12	38,280 kB
2048	49.6	1.04 s	25.07 ms	39.89	74,668 kB
4096	53.3	2.35 s	48.63 ms	20.56	148,936 kB
8192	57.0	7.27 s	96.67 ms	10.34	302,616 kB
16384	60.7	14.41 s	205.35 ms	4.87	618,448 kB

as a subroutine for Peikert's sampler [Pei10] for sampling from $D_{(g),\sigma',0}$ with $g \in \mathbb{Z}[x]/(x^N+1)$ for N a power of two. To this end, we adapted the implementation of this sampler from [ACLL15] where we swap out the sampler from the dgs library [Alb14] (implementing rejection sampling and [DDLL13]) used in [ACLL15] with our sampler for sampling for $D_{\mathbb{Z},\sigma,c}$. Note that sampling from $D_{(g),\sigma',0}$ is more involved and thus slower than sampling from $D_{\mathbb{Z}^N,\sigma',0}$. That is, to sample from $D_{(g),\sigma',0}$, [ACLL15] first computes an approximate square root of $\Sigma_2 = \sigma'^2 \cdot g^{-T} \cdot g^{-1} - r^2$ with $r = 2 \cdot \lceil \sqrt{\log N} \rceil$. Then, given an approximation $\sqrt{\Sigma_2}'$ of $\sqrt{\Sigma_2}$ it samples a vector $x \leftarrow_\$ \mathbb{R}^N$ from a standard normal distribution and interpret it as a polynomial in $\mathbb{Q}[X]/(x^N+1)$; computes $y = \sqrt{\Sigma_2}' \cdot x$ in $\mathbb{Q}[X]/(x^N+1)$ and returns $g \cdot (\lfloor y \rceil_r)$, where $\lfloor y \rceil_r$ denotes sampling a vector in $\mathbb{Z}^N$ where the i-th component follows $D_{\mathbb{Z},r,y_i}$. Thus, implementing Peikert's sampler requires sampling from $D_{\mathbb{Z},r,y_i}$ for changing centers y_i and sampling from a standard normal distribution. We give experimental results in the table 2.2, indicating that our sampler increases the rate by a factor ≈ 3.

2.4.2.2. Double-Precision Twin-CDT

The double precision Twin-CDT algorithm was implemented with standard IEEE 754 double precision floating-point arithmetic and binary search executed byte by byte. The implementation uses a

Table 2.3. – Performance of the GPV signature generation. All timings are on a 2.90GHz Intel(R) Core(R) i5–4210H and use one core.

Security parameter λ	**80**	**192**
Polynomial degree N	512	1024
Number of samples per signature	1024	2048
Gaussian parameter s	≈ 0.9	≈ 2.4
Signature generation (rejection sampling)	65.5 ms	168.9 ms
Signature generation (Twin-CDT)	8.3 ms	42.2 ms

lookup table of size $\ell = 2^8$ as described in section 2.4.1.1, does not include the Taylor approximation described in section 2.3.2.2 and is not constant-time. We tested the performance of our double precision Twin-CDT algorithm by using it as subroutine for GPV signature generation, as well as for a general set of parameters.

GPV performance. In [DLP14], a particular distribution over NTRU [HPS98] lattices is used to increase the efficiency of GPV-based schemes. A lattice-based IBE scheme, whose the key extraction algorithm consists of a GPV signature generation, is also presented with concrete parameters. Its implementation[6] is considered the current reference for evaluating GPV performance (whether it is used for key extraction or signature generation). We adapted this implementation, which initially used rejection sampling, with our double precision Twin-CDT implementation. Note that the tested rejection sampler used the MPFR [FHL+07] library as well as Salsa20 as the pseudorandom number generator and that the Twin-CDT algorithm uses two precomputed CDTs (i.e. $n = 2$) for each Gaussian parameter s_i with $i \in \{0, \ldots, 2N\}$, where N is the NTRU polynomial degree. We give experimental results in the table 2.3, indicating that our approach increases the GPV signature rate (or similarly the key extraction rate) by a factor 4 to 8 depending on the security parameter.

Fixed gaussian parameter with varying center. In this setting the Gaussian parameter is fixed but the center may vary. We present, in the table 2.4, some experimental results. Our implementation is clearly non-optimized, which leads us to believe that one can improve its performance with a reasonable effort. In accordance with sections 2.3.1.3 and 2.4.1.1, the memory used is equal to $128\lceil s\sqrt{\lambda}/2\rceil n + 64$ bits. When referring to other implementations the reader has to be aware of the two common definitions of $\rho(\cdot)$ in the literature, some use the Gaussian parameter s as in this paper and others use $\sigma = s\sqrt{2\pi}$. Note that the improvement gained by using more memory deteriorates in our implementation up to the point where using more memory does not decrease significantly more the running time. To obtain a more flexible time-memory trade-off one can use the Taylor approximation described in section 2.3.2.2.

References

[ACLL15] Martin R. Albrecht, Catalin Cocis, Fabien Laguillaumie, and Adeline Langlois. *Implementing Candidate Graded Encoding Schemes from Ideal Lattices*. In: *ASIACRYPT 2015,*

[6] `https://github.com/tprest/Lattice-IBE`

Part II. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 752–775 (cit. on p. 53).

[Alb14] Martin R. Albrecht. *dgs — Discrete Gaussians over the Integers*. `https://bitbucket.org/malb/dgs`. 2014 (cit. on pp. 51–53).

[BCG+14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. *Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers*. In: *SAC 2013*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. LNCS. Springer, Heidelberg, Aug. 2014, pp. 402–417 (cit. on pp. 27, 32).

[Ber08] Daniel J. Bernstein. "New Stream Cipher Designs". In: ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer-Verlag, 2008. Chap. The Salsa20 Family of Stream Ciphers, pp. 84–97 (cit. on pp. 28, 51).

[BHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. *Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme*. In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. 2016, pp. 323–345 (cit. on p. 51).

[BLL+15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. *Improved Security Proofs in Lattice-Based Cryptography: Using the Rényi Divergence Rather Than the Statistical Distance*. In: *ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. Springer, Heidelberg, Nov. 2015, pp. 3–24 (cit. on pp. xix, xxii, 29, 30, 41, 44, 93).

[Bre+06] Richard P Brent et al. *Fast algorithms for high-precision computation of elementary functions*. In: *Proceedings of 7th conference on real numbers and computers (RNC 7)*. 2006, pp. 7–8 (cit. on p. 36).

[CRVV15] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. *Efficient software implementation of ring-LWE encryption*. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2015, pp. 339–344 (cit. on p. 49).

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. *Lattice Signatures and Bimodal Gaussians*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 40–56 (cit. on pp. 13, 27, 32, 53).

[Dev86] L. Devroye. *Non-uniform random variate generation*. Springer-Verlag, 1986 (cit. on pp. 28, 32).

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. "Sampling from discrete Gaussians for lattice-based cryptography on a constrained device". In: *Applicable Algebra in Engineering, Communication and Computing* 25.3 (2014), pp. 159–180 (cit. on pp. 27, 33).

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. *Efficient Identity-Based Encryption over NTRU Lattices*. In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 22–41 (cit. on pp. xiii, 54, 62, 63, 65, 66, 70).

[DN12a] Léo Ducas and Phong Q. Nguyen. *Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 415–432 (cit. on pp. xix, 32, 33, 36, 37, 40).

[Duc13] Léo Ducas. *Lattice Based Signatures: Attacks, Analysis and Optimization*. Ph.D. Thesis. 2013 (cit. on p. 35).

[EH14] Tim van Erven and Peter Harremoës. "Rényi Divergence and Kullback-Leibler Divergence". In: *IEEE Trans. Information Theory* 60.7 (2014), pp. 3797–3820 (cit. on pp. xix, 30).

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. "MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding". In: *ACM Trans. Math. Softw.* 33.2 (June 2007) (cit. on pp. 29, 51, 54).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. *Trapdoors for hard lattices and new cryptographic constructions*. In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206 (cit. on pp. xii, xiii, 9, 13, 14, 31, 62–65).

[Gt15] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 6.0.1. `http://gmplib.org/`. 2015 (cit. on p. 51).

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A Ring-Based Public Key Cryptosystem*. In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. 1998, pp. 267–288 (cit. on pp. 7, 10, 11, 54, 65).

[Kar16] Charles F. F. Karney. "Sampling Exactly from the Normal Distribution". In: *ACM Trans. Math. Softw.* 42.1 (Jan. 2016), 3:1–3:14 (cit. on pp. 27, 32).

[KY76] Donald E. Knuth and Andrew C. Yao. "The Complexity of Nonuniform Random Number Generation". In: *Algorithms and Complexity: New Directions and Recent Results*. Ed. by J. F. Traub. New York: Academic Press, 1976 (cit. on p. 32).

[MR07] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM J. Comput.* 37.1 (Apr. 2007), pp. 267–302 (cit. on pp. 12, 31, 33).

[MT84] G. Marsaglia and W. W. Tsang. "A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions". In: *SIAM J. Sci. Stat. Comput.* 5 (1984), pp. 349–359 (cit. on p. 32).

[MW17] Daniele Micciancio and Michael Walter. *Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time*. In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 455–485 (cit. on pp. xviii, 27, 29, 30, 33, 69, 71).

[Neu51] John von Neumann. "Various Techniques Used in Connection with Random Digits". In: *J. Res. Nat. Bur. Stand.* 12 (1951), pp. 36–38 (cit. on p. 32).

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. *Enhanced Lattice-Based Signatures on Reconfigurable Hardware*. In: *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. Ed. by Lejla Batina and Matthew Robshaw. Springer Berlin Heidelberg, 2014, pp. 353–370 (cit. on pp. xviii, xxi, xxii, 30, 40, 43, 44).

[Pei10] Chris Peikert. *An Efficient and Parallel Gaussian Sampler for Lattices*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 80–97 (cit. on pp. xiv, xix, xxi, 14, 27, 33, 40, 41, 43, 53, 63, 67).

[Pre17] Thomas Prest. *Sharper Bounds in Lattice-Based Cryptography Using the Rényi Divergence*. In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Heidelberg, Dec. 2017, pp. 347–374 (cit. on pp. xviii, 30, 40, 68, 69, 92).

[PS08] Xavier Pujol and Damien Stehlé. *Rigorous and Efficient Short Lattice Vectors Enumeration*. In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 390–405 (cit. on p. 35).

[Saa16] Markku-Juhani O. Saarinen. *Arithmetic Coding and Blinding Countermeasures for Lattice Signatures: Engineering a Side-Channel Resistant Post-Quantum Signature Scheme with Compact Signatures*. Cryptology ePrint Archive, Report 2016/276. `http://eprint.iacr.org/2016/276`. 2016 (cit. on p. 32).

[Von51] John Von Neumann. "The general and logical theory of automata". In: *Cerebral mechanisms in behavior* 1.41 (1951), pp. 1–2 (cit. on p. 32).

[Wal74] A. J. Walker. "New fast method for generating discrete random numbers with arbitrary frequency distributions". In: *Electronics Letters* 10 (Apr. 1974), 127–128(1) (cit. on p. 32).

[ZWR+16] Samee Zahur, Xiao Shaun Wang, Mariana Raykova, Adria Gascón, Jack Doerner, David Evans, and Jonathan Katz. *Revisiting Square-Root ORAM: Efficient Random Access in Multi-party Computation*. In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. 2016, pp. 218–234 (cit. on p. 51).

Table 2.4. – Performance of the Twin-CDT algorithm in double precision. All timings are on a 2.90GHz Intel(R) Core(R) i5–4210H and use one core. The trade-off parameter n is the number of centers for which the CDTs are precomputed, s is the Gaussian parameter and λ is the security parameter, which determines the tailcut parameter $t = \sqrt{\lambda}/2$

s	λ	n	**Samples per second**	**Memory used**
2	128	3	35 226 936	1.4 KB
		10	43 123 782	4.6 KB
		100	52 777 917	46 KB
	256	3	35 086 446	1.6 KB
		10	43 104 933	5.2 KB
		100	52 935 721	52 KB
4	128	3	31 422 214	1.9 KB
		10	38 973 842	6.3 KB
		100	47 707 711	63 KB
	256	3	31 660 993	2.3 KB
		10	38 566 583	7.7 KB
		100	46 910 085	77 KB
8	128	3	26 036 426	3 KB
		10	36 814 796	10 KB
		100	43 857 717	100 KB
	256	3	25 932 004	3.9 KB
		10	36 123 197	13 KB
		100	42 683 912	128 KB
16	128	3	21 610 550	5.2 KB
		10	25 983 612	17 KB
		100	36 111 573	172 KB
	256	3	21 142 623	6.9 KB
		10	25 957 210	23 KB
		100	36 210 391	231 KB
32	128	3	23 742 886	9.5 KB
		10	27 456 854	32 KB
		100	29 420 447	318 KB
	256	3	24 402 531	13 KB
		10	27 238 625	44 KB
		100	29 434 654	436 KB

It's the ship that made the Kessel run in less than twelve parsecs. I've outrun Imperial starships. Not the local bulk cruisers, mind you. I'm talking about the big Corellian ships, now. She's fast enough for you, old man.

Star Wars (Han Solo) – George Lucas

Falcon, A New Compact Signature Scheme over NTRU

3

Falcon is a lattice-based signature scheme. It stands for the following acronym:

Fast Fourier lattice-based compact signatures over NTRU

This chapter is a detailed version of the Falcon specification [FHK+17] coauthored with Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Gregor Seiler, William Whyte, Zhenfei Zhang and submitted to NIST Post-Quantum Cryptography Project on November 30th, 2017.

Contents

The high-level design of Falcon is simple: we instantiate the theoretical framework described by Gentry, Peikert and Vaikuntanathan [GPV08] for constructing hash-and-sign lattice-based signature schemes. This framework requires two ingredients:

- A class of cryptographic lattices. We chose the class of NTRU lattices.
- A trapdoor sampler. We rely on a new technique which we call fast Fourier sampling.

In a nutshell, the Falcon signature scheme may therefore be described as follows:

Falcon = GPV framework + NTRU lattices + Fast Fourier sampling

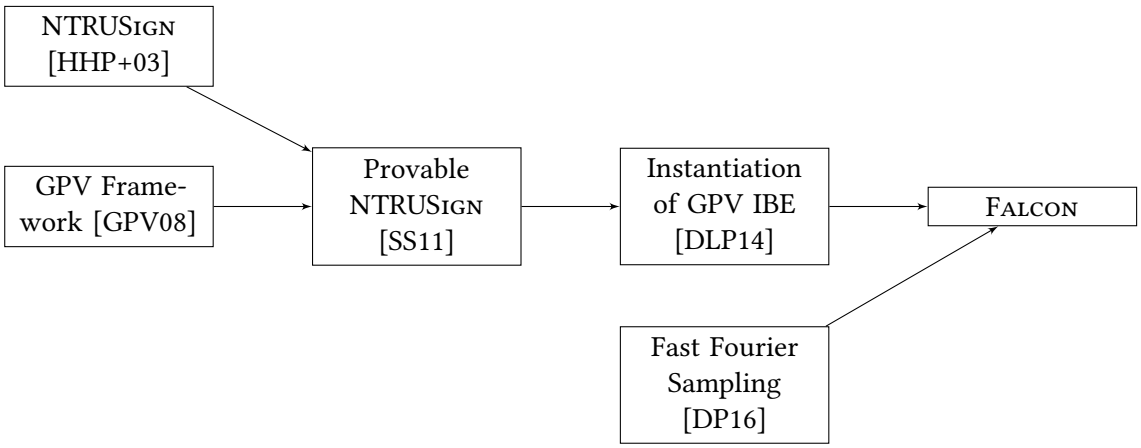


Figure 3.1. – The genealogic tree of Falcon

Falcon is the product of many years of work, not only by the authors but also by others. The first work is the signature scheme NTRUSign [HHP+03] by Hoffstein *et al.*, which was the first, along with GGH [GGH97], to propose lattice-based signatures. The use of NTRU lattices by NTRUSign allows it to be very compact. However, both had a flow in the deterministic signing procedure which led to devastating key-recovery attacks [NR06; DN12b].

At STOC 2008, Gentry, Peikert and Vaikuntanathan [GPV08] proposed a method which not only corrected the flawed signing procedure but, even better, did it in a provably secure way. The result was a generic framework (the GPV framework) for building secure hash-and-sign lattice-based signature schemes.

The next step towards Falcon was the work of Stehlé and Steinfeld [SS11], who combined the GPV framework with NTRU lattices. The result could be called – somewhat surprisingly – a provably secure NTRUSign.

In a more practical work, Ducas *et al.* [DLP14] proposed a practical instantiation and implementation of the IBE part of the GPV framework over NTRU lattices. This IBE can be converted in a straightforward manner into a signature scheme. However, doing this would have resulted in a signing time in $O(n^2)$.

To address the issue of a slow signing time, Ducas and Prest [DP16] proposed a new algorithm running in time $O(n \log n)$. However, how to practically instantiate this algorithm remained an open question.

Falcon builds on these works to propose a practical lattice-based hash-and-sign scheme. The figure 3.1 shows the genealogic tree of Falcon.

The design rationale of Falcon stems from a simple observation: when switching from signatures based on RSA or the discrete logarithm to post-quantum signatures, communication complexity is likely going to be a larger problem than speed. Indeed, many post-quantum schemes have a simple algebraic description which makes them fast, but they always require either larger keys than pre-quantum schemes, larger signatures, or both.

We expect such performance issues will hinder transition from pre-quantum to post-quantum schemes. Hence our leading design principle, which underlied most of our decisions, was to minimize the following quantity:

$$|\mathsf{pk}|+|\mathsf{sig}| = \text{(bitsize of the public key)} + \text{(bitsize of a signature)}.$$

This led us to consider lattice-based signatures, which manage to keep both $|\mathsf{pk}|$ and $|\mathsf{sig}|$ rather small, especially for structured lattices. When it comes to lattice-based signatures, there are essentially two paradigms: Fiat-Shamir or hash-and-sign.

Both paradigms achieve comparable levels of compactness, but hash-and-sign have interesting properties: the GPV framework [GPV08], which describes how to obtain hash-and-sign lattice-based signature schemes, is secure in the classical and quantum oracle models [GPV08; BDF+11]. In addition, it enjoys message-recovery capabilities [PLP16]. So we chose this framework. We detail this choice and its implications in section 3.1.

The next step was to choose a class of cryptographic lattices to instantiate this framework. A close to optimal choice with respect to our main design principle – compactness – is NTRU lattices: as sketched in [DLP14], they allow to obtain a rather compact instantiation of the GPV framework. In addition, they come with a ring structure which speeds up many operations by two orders of magnitude (e.g. a factor $O(n/\log n)$ for signature verification). We detail this choice and its implications in section 3.1.1.

The last step was the trapdoor sampler. We devised a new trapdoor sampler which is asymptotically as fast as the fastest generic trapdoor sampler [Pei10] and provides the same level of security as the most secure sampler [Kle00]. We detail this choice and its implications in section 3.1.2.

3.1. Instantiate the GPV Framework over NTRU Lattices

In 2008, Gentry, Peikert and Vaikuntanathan [GPV08] established a framework for obtaining secure lattice-based signatures. At a very high level, this framework may be described as follows:

- The public key contains a full-rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ (with $m > n$) generating a q-ary lattice Λ.
- The private key contains a matrix $\mathbf{B} \in \mathbb{Z}_q^{m\times m}$ generating $\Lambda_q^\perp$, where $\Lambda_q^\perp$ denotes the lattice orthogonal to Λ modulo q: for any $\mathbf{x} \in \Lambda$ and $\mathbf{y} \in \Lambda_q^\perp$, we have $\langle \mathbf{x}, \mathbf{y}\rangle = 0 \bmod q$. Equivalently, the rows of $\mathbf{A}$ and $\mathbf{B}$ are pairwise orthogonal: $\mathbf{B} \times \mathbf{A}^\mathsf{t} = \mathbf{0}$.
- Given a message m, a signature of m is a short value $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{s}\mathbf{A}^\mathsf{t} = H(\mathsf{m})$, where $H : \{0,1\}^* \to \mathbb{Z}_q^n$ is a hash function. Given $\mathbf{A}$, verifying that $\mathbf{s}$ is a valid signature is straightforward: it only requires to check that $\mathbf{s}$ is indeed short and verifies $\mathbf{s}\mathbf{A}^\mathsf{t} = H(\mathsf{m})$.
- Computing a valid signature is more delicate. First, an arbitrary preimage $\mathbf{c}_0 \in \mathbb{Z}_q^m$ is computed, which verifies $\mathbf{c}_0\mathbf{A}^\mathsf{t} = \mathbf{c}$. As $\mathbf{c}_0$ is not required to be short and $m \geq n$, this can simply be done through standard linear algebra. $\mathbf{B}$ is then used in order to compute

a vector $\mathbf{v} \in \Lambda_q^\perp$ close to $\mathbf{c}_0$. The difference $\mathbf{s} = \mathbf{c}_0 - \mathbf{v}$ is a valid signature: indeed, $\mathbf{s}\mathbf{A}^t = \mathbf{c}_0\mathbf{A}^t - \mathbf{v}\mathbf{A}^t = \mathbf{c} - \mathbf{0} = \mathbf{c}$, and if $\mathbf{c}_0$ and $\mathbf{v}$ are close enough, then $\mathbf{s}$ is short.

In this abstract form, this description of a signature scheme is not specific to the GPV framework: it was first instantiated in the GGH [GGH97] and NTRUSign [HHP+03] signature schemes. However, GGH and NTRUSign suffer of total break attacks, whereas the GPV framework is proven to be secure in the classical and quantum random oracle models assuming the hardness of SIS for some parameters. The reason behind this is somewhat subtle, and lies in the fact that GGH/NTRUSign and the GPV framework have radically different ways of computing $\mathbf{v}$ in the signing procedure.

Computing $\mathbf{v}$ in GGH and NTRUSign. In GGH and NTRUSign, $\mathbf{v}$ is computed using an algorithm called the round-off algorithm and first formalized by Babai [Bab85; Bab86]. In this deterministic algorithm, $\mathbf{c}_0$ is first expressed as a real linear combination of the rows of $\mathbf{B}$, the vector of these real coordinates is then rounded coefficient-wise and multiplied again by $\mathbf{B}$: in a nutshell, $\mathbf{v} \leftarrow \left\lfloor \mathbf{c}_0\mathbf{B}^{-1} \right\rceil \mathbf{B}$, where $\lfloor \cdot \rceil$ denotes coefficient-wise rounding. At the end of the procedure, $\mathbf{s} = \mathbf{v} - \mathbf{c}_0$ is guaranteed to lie in the parallelepiped $[-1, 1]^m \times \mathbf{B}$, which allows to tightly bound the norm $\|\mathbf{s}\|$.

The problem with this approach is that each signature $\mathbf{s}$ lies in $[-1, 1]^m \times \mathbf{B}$, and therefore each $\mathbf{s}$ leaks a little information about the basis $\mathbf{B}$. This fact was successfully exploited by several attacks [NR06; DN12b] which led to a total break on the schemes.

Computing $\mathbf{v}$ in the GPV framework. A major contribution of [GPV08], which is also the key difference between the GPV framework and GGH/NTRUSign, is the way $\mathbf{v}$ is computed. Instead of the round-off algorithm, the GPV framework relies on a randomized variant by [Kle00] of the nearest plane algorithm, also formalized by Babai. Just as for the round-off algorithm, using the nearest plane algorithm would have leaked the secret basis $\mathbf{B}$ and resulted in a total break of the scheme. However, Klein's algorithm prevents this: it is randomized in a way such that for a given m, $\mathbf{s}$ is sampled according to a spherical Gaussian distribution over the shifted lattice $\mathbf{c}_0 + \Lambda_q^\perp$. This method is not only impervious to the attacks described hereabove, but is also proven to leak no information about the basis $\mathbf{B}$. Klein's algorithm was in fact the first of a family of algorithms called trapdoor samplers. More details about trapdoor samplers are given in section 3.1.2.

3.1.0.1. Features and instantiation of the GPV framework

The topic of this section is to make explicit a few aspects and features of the GPV framework.

Security in the classical and quantum oracle models. In the original paper [GPV08], the GPV framework has been proven to be secure in the random oracle model under the SIS assumption. In our case, we use NTRU lattices so we need to adapt the proof for a "NTRU-SIS" assumption, but this adaptation is straightforward. In addition, the GPV framework has also been proven to be secure in the quantum oracle model [BDF+11].

Signatures with message recovery. In [PLP16], it has been shown that a preliminary version of Falcon can be instantiated in message-recovery mode: the message m can be recovered from the signature sig. It requires to make the signature twice longer, but it allows to entirely recover a message which size is a bit less than half the size of the original signature. In situations where we can apply it, it makes Falcon even more competitive from a compactness viewpoint.

Identity-based encryption. FALCON can be turned into an identity-based encryption scheme. This is described in [DLP14]. However, this requires de-randomizing the signature procedure (see the paragraph "Statefulness, de-randomization or hash randomization").

3.1.0.2. Statefulness, de-randomization or hash randomization

In the GPV framework, two different signatures $\mathbf{s}, \mathbf{s}'$ of a same hash $H(\mathsf{m})$ can never be made public simultaneously, because doing so breaks the security proof [GPV08, Section 6.1].

Statefulness. A first solution proposed in [GPV08, Section 6.1] is to make the scheme stateful by maintaining a list of the signed messages and of their signatures. However, maintaining such a state poses a number of operational issues, so we do not consider it as a credible solution.

De-randomization. A second possibility proposed by [GPV08] is to de-randomize the signing procedure. However, this raises another issue as pseudorandomness would need to be generated in a consistent way over all the implementations (it is not uncommon to have a same signing key used in different devices). While this solution can be applied in a few specific usecases, we do not consider it for the FALCON signature scheme.

Hash randomization. A third solution is to prepend a salt $\mathsf{r} \in \{0,1\}^k$ to the message m before hashing it. Provided that k is large enough, this effectively prevents collisions from occuring with non-negligible probability. From an operational perspective, this solution is the easiest to apply, and it is still covered by the security proof of the GPV framework (see [GPV08, Section 6.2]). For a given security level λ and up to q_s signature queries, taking $k = \lambda + \log_2(q_s)$ is enough to guarantee that the probability of collision is less than $q_s \cdot 2^{-\lambda}$.

Out of the three solutions, FALCON opts for hash randomization: a salt $\mathsf{r} \in \{0,1\}^{320}$ is randomly generated and prepended to the message before hashing it. The bitsize 320 is equal to $\lambda + \log_2(q_s)$ for $\lambda = 256$ the highest security level required by NIST, and $q_s = 2^{64}$ the maximal number of signature which may be queried from a single signer. This size is actually overkill for security levels $\lambda < 256$, but fixing a single size across all the security levels makes things easier from an API perspective: for example, one can hash a message without knowing the security level of the private signing key.

3.1.1. NTRU Lattices

The first choice when instantiating the GPV framework is the class of lattices to use. The design rationale obviously plays a large part in this. Indeed, if emphasis is placed on security without compromise, then the logical choice is to use standard lattices without any additional structure, as was done e.g. in the key-exchange scheme FRODO [BCD+16].

Our main design principle is compactness. For this reason, FALCON rely on the class of NTRU lattices, introduced by Hoffstein, Pipher and Silverman [HPS98]; they come with an additional ring structure which not only does allow to reduce the public keys' size by a factor $O(n)$, but also speeds up many computations by a factor at least $O(n/\log n)$. Even in the broader class of lattices over rings, NTRU lattices are among the most compacts: the public key can reduced to a single polynomial $h \in \mathbb{Z}_q[x]$ of degree at most $n-1$. In doing this we follow the idea of Stehlé and Steinfeld [SS11], which have shown that the GPV framework can be used in conjunction with NTRU lattices in a provably secure way.

Compactness, however, would be useless without security. From this perspective, NTRU lattices also have reasons to inspire confidence as they have resisted extensive cryptanalysis for about two decades, and we parameterize them in a way which we believe makes them even more resistant.

3.1.1.1. Introduction to NTRU lattices

Let $\phi \in \mathbb{Z}[x]$ be a monic polynomial, and $q \in \mathbb{N}^\star$. A set of NTRU secrets consists of four polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ which verify the NTRU equation:

$$fG - gF = q \mod \phi \tag{3.1}$$

Provided that f is invertible modulo q, we can define the polynomial $h \leftarrow g \cdot f^{-1} \bmod q$.

Typically, h will be a public key, whereas f, g, F, G will be secret keys. Indeed, one can check that the matrices $\left[\begin{array}{c|c} 1 & h \\ \hline 0 & q \end{array}\right]$ and $\left[\begin{array}{c|c} f & g \\ \hline F & G \end{array}\right]$ generate the same lattice, but the first matrix contains two large polynomials (h and q), whereas the second matrix contains only small polynomials, which allows to solve problems as illustrated in section 3.1. If f, g are generated with enough entropy, then h will look pseudo-random [SS11]. However in practice, even when f, g are quite small, it remains hard to find small polynomials f', g' such that $h = g' \cdot (f')^{-1} \bmod q$. The hardness of this problem constitutes the NTRU assumption.

3.1.1.2. Instantiation with the GPV framework

We now instantiate the GPV framework described in section 3.1 over NTRU lattices:

- The public basis is $\mathbf{A} = \left[\begin{array}{c|c} 1 & h^\star \end{array}\right]$, but this is equivalent to knowing h.
- The secret basis is
$$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right] \tag{3.2}$$
One can check that the matrices $\mathbf{A}$ and $\mathbf{B}$ are indeed orthogonal: $\mathbf{B} \times \mathbf{A}^\star = 0 \bmod q$.
- The signature of a message m consists of a salt r plus a pair of polynomials (s_1, s_2) such that $s_1 + s_2 h = H(\mathsf{r} \| \mathsf{m})$. We note that since s_1 is completely determined by m, r and s_2, there is no need to send it: the signature can simply be (r, s_2).

3.1.1.3. Choosing optimal parameters

Our trapdoor sampler samples signatures of norm essentially proportional to $\|\mathbf{B}\|_{\text{GS}}$, where $\|\mathbf{B}\|_{\text{GS}}$ denotes the Gram-Schmidt norm of $\mathbf{B}$.

Previous works ([DLP14] and [Pre15, Sections 6.4.1 and 6.5.1]) have provided heuristic and experimental evidence that in practice, $\|\mathbf{B}\|_{\text{GS}}$ is minimized for $\|(f, g)\| \approx 1.17\sqrt{q}$. Therefore, we generate f, g as discrete Gaussians in $\mathbb{Z}[x]/(\phi)$ centered in 0, so that the expected value of $\|(f, g)\|$ is about $1.17\sqrt{q}$. Once this is done, very efficient ways to compute $\|\mathbf{B}\|_{\text{GS}}$ are known, and if this value is more than $1.17\sqrt{q}$, new polynomials f, g's are regenerated and the procedure starts over.

Quasi-optimality. The bound $\|\mathbf{B}\|_{\text{GS}} \leq 1.17\sqrt{q}$ that we reach in practice is within a factor 1.17 of the theoretic lower bound for $\|\mathbf{B}\|_{\text{GS}}$. Indeed, for any $\mathbf{B}$ of the form given in equation 3.2 with f, g, F, G verifying the equation 3.1, we have $\det(\mathbf{B}) = fG - gF = q$. So $\sqrt{q}$ is a theoretic lower bound of $\|\mathbf{B}\|_{\text{GS}}$.

3.1.2. Fast Fourier Sampling

The second choice when instantiating the GPV framework is the trapdoor sampler. A trapdoor sampler takes as input a matrix $\mathbf{A}$, a trapdoor T, a target $\mathbf{c}$ and outputs a short vector $\mathbf{s}$ such that $\mathbf{s}^t\mathbf{A} = \mathbf{c} \bmod q$. With the notations of section 3.1, this is equivalent to finding $\mathbf{v} \in \Lambda_q^\perp$ close to $\mathbf{c}_0$, so we may indifferently refer by the term "trapdoor samplers" to algorithms which perform one task or the other.

We now list the existing trapdoor samplers, their advantages and limitations. Obviously, being efficient is important for a trapdoor sampler. However, an equally important metric is the "quality" of the sampler: the shorter the vector $\mathbf{s}$ is (or equivalently, the closer $\mathbf{v}$ is to $\mathbf{c}_0$), the more secure this sampler will be.

1. Klein's algorithm [Kle00] takes as a trapdoor the matrix $\mathbf{B}$. It outputs vectors $\mathbf{s}$ of norm proportional to $\|\mathbf{B}\|_{\text{GS}}$, which is short and therefore good for security. On the downside, its time and space complexity are in $O(m^2)$.

2. Just like Klein's algorithm is a randomized version of the nearest plane algorithm, Peikert proposed a randomized version of the round-off algorithm [Pei10]. The good part about it is that when $\mathbf{B}$ has a structure over rings – as in our case – then it can be made to run in time and space $O(m \log m)$. However, it outputs vectors of norm proportional to the spectral norm $\|\mathbf{B}\|_2$ of $\mathbf{B}$. This is larger than what we get with Klein's algorithm, and therefore it is worse security-wise.

3. Micciancio and Peikert [MP12] proposed a novel approach in which $\mathbf{A}$ and its trapdoor are constructed in a way which allows simple and efficient trapdoor sampling. This approach was generalized in [LW15]. Unfortunately, it is not straightforwardly compatible with NTRU lattices and whether we can reach the same level of compactness as with NTRU lattices is unclear.

4. Ducas and Prest [DP16] proposed a variant of Babai's nearest plane algorithm for lattices over rings. It proceeds in a recursive way which is very similar to the fast Fourier transform, and for this reason they dubbed it "fast Fourier nearest plane". This algorithm can be randomized as well: it results in a trapdoor sampler which combines the quality of Klein's algorithm, the efficiency of Peikert's and can be used over NTRU lattices.

Of the four approaches we just described, it seems clear to us that a randomized variant of the fast Fourier nearest plane [DP16] is the most adequate choice given our design rationale and our previous design choices (NTRU lattices). For this reason, it is the trapdoor sampler used in Falcon.

Sampler	Fast	Short output $\mathbf{s}$	NTRU-friendly
Klein [Kle00]	**No**	**Yes**	**Yes**
Peikert [Pei10]	**Yes**	**No**	**Yes**
Micciancio-Peikert [MP12]	**Yes**	**Yes**	**No**
Ducas-Prest [DP16]	**Yes**	**Yes**	**Yes**

Table 3.1. – Comparison of the different trapdoor samplers

Choosing the standard deviation. When using a trapdoor sampler, an important parameter to set is the standard deviation σ. If it is too low, then it is no longer guaranteed that the sampler not leak the secret basis (and indeed, for all known samplers, a value $\sigma = 0$ opens the door to learning attacks à la [NR06; DN12b]). But if it is too high, the sampler does not return optimally short vectors and the scheme is not as secure as it could be. So there is a compromise to be found.

Our fast Fourier sampler shares many similarities with Klein's sampler, including the optimal value for σ (i.e. the shortest which is known not to leak the secret basis). According to [Pre17], it is sufficient for the security level and number of queries set by NIST to take $\sigma \leq 1.312 \|\mathbf{B}\|_{\text{GS}}$, which in our case translates to $\sigma \leq 1.55\sqrt{q}$.

3.1.3. Security

3.1.3.1. Known Attacks

Key Recovery. The most efficient attacks come from lattice reduction. We start by considering the lattice $(\mathbb{Z}[x]/(\phi))^2 \left[\begin{array}{c|c} 0 & q \\ \hline 1 & h \end{array}\right]$. After using lattice reduction on this basis, we enumerate all lattice points in a ball of radius $\sqrt{2n}\sigma'$, centered on the origin. With significant probability, we are therefore able to find $\left[\begin{array}{c|c} g & f \end{array}\right]$. If we use a block-size of B, enumeration takes negligible time if the $2n - B$th Gram-Schmidt norm is larger than $0.75\sqrt{B}\sigma'$. For the best known lattice reduction algorithm, DBKZ [MW16], it is

$$(\frac{B}{2\pi\text{e}})^{(1-n/B)}\sqrt{q}.$$

It is then easy to deduce B, and to show that $B = n + o(n)$. This gives $B = 652$ when $n = 768$ and $B = 921$ when $n = 1024$. The security implied is detailed in the following table, using the methodology of New Hope [ADPS16].

n	B	Classical	Quantum
512	392	114	103
768	652	195	172
1024	921	263	230

Forging a Signature. Forging a signature can be perfomed by finding a lattice point at distance bounded by β from a random point, in the same lattice as above. This task is also eased by first carrying out lattice reduction on the original basis. One possibility is to enumerate all lattice points in a ball of radius $\sqrt{\frac{nq}{\pi\text{e}}}$. As this ball is larger than the one of the previous attack, it would be slower. It may seem as if it would be much smaller than the previous attack due to a factor $\Theta(\sqrt{n})$ in the radius. It is not the case, since the lattice has an (almost) orthogonal basis, which implies there are few ($2^{o(n)}$) points at distance in $o(\sqrt{n})$. This implies that the proposed method essentially starts by recovering the secret key, so that it is slower than the previous algorithm. Also, embedding the point in the lattice does not help: the distance to the lattice is $\Theta(\sqrt{n})$ greater than the shortest non-zero point.

Combinatorial attack. If we were to choose $q = O(n)$, the size of the coefficients would be constant. Then, Kirchner-Fouque [KF15] BKW variant would run in time $2^{n/((2+o(1))\log\log n)}$ to recover the key, i.e. asymptotically faster than the previous algorithms. It indicates that the most compact scheme uses $q = n^{1+\epsilon+o(1)}$ for some $\epsilon > 0$. However, since n is not huge, our moderate q

is enough to make this attack irrelevant. Indeed, even assuming that nearest neighbor search runs in constant time and other optimistic assumptions, the best combinatorial attack runs in time 2^{135} for $n = 512$.

Hybrid attack. The hybrid attack [How07] combines a meet-in-the-middle algorithm and the key recovery algorithm. It was used with great effect against NTRU, due to its choice of *sparse* polynomials. This is however not the case here, so that its impact is much more modest, and counterbalanced by the lack of sieve-enumeration.

Dense, high rank sublattice. Recent works [ABD16; CJL16; KF17] have shown that when f, g are extremely small compared to q, it is easy to attack cryptographic schemes based on NTRU lattices. To the contrary, in FALCON we take f, g to be not too small while q is hardly large: a side-effect is that this makes our scheme impervious to the so-called "overstretched NTRU" attacks. In particular, even if f, g were taken to be binary, we would have to select $q > n^{2.83}$ for this property to be useful for cryptanalysis. Our large margin should allow even significant improvements of this algorithm to be irrelevant to our case.

Algebraic attacks. While there is a rich algebraic structure in FALCON, there is no known way to improve all the algorithms previously mentioned with respect to their general lattice equivalent by more than a factor n^2. However, there exist efficient algorithms for finding not-so-small elements in *ideals* of $\mathbb{Z}[x]/(\phi)$ [CDW17].

3.1.3.2. Precision of the Floating-Point Arithmetic

Trapdoor samplers usually require the use of floating-point arithmetics, and our fast Fourier sampler is no exception. This naturally raises the question of the precision required to claim meaningful security bounds. A naive analysis would require a precision of $O(\lambda)$ bits (nonwithstanding logarithmic factors), but this would result in a substantially slower signature generation procedure.

In order to analyze the required precision, we use a Rényi divergence argument. As in [MW17], we denote by $a \lesssim b$ the fact that $a \leq b + o(b)$, which allows to discard negligible factors in a rigorous way. Our fast Fourier sampler is a recursive algorithm which relies on $2n$ discrete samplers $D_{\mathbb{Z},c_j,\sigma_j}$. We suppose that the values c_j (resp. σ_j) are known with an *absolute* error (resp. *relative* error) at most δ_c (resp. δ_σ) and denote by $\mathcal{D}$ (resp. $\bar{\mathcal{D}}$) the output distribution of our sampler with infinite (resp. finite) precision. We can then re-use the precision analysis of Klein's sampler in [Pre17, Section 4.5]. For any output of our sampler with non-negligible probability, in the worst case:

$$\left|\log\left(\frac{\bar{\mathcal{D}}(\mathbf{z})}{\mathcal{D}(\mathbf{z})}\right)\right| \lesssim 2n\left[\frac{\sqrt{154}}{1.312}\delta_c + (2\pi+1)\delta_\sigma\right] \leq 20n(\delta_c+\delta_\sigma) \tag{3.3}$$

In the average case, the value $2n$ in equation 3.3 can be replaced with $\sqrt{2}n$. Following the security arguments of [Pre17, Section 3.3], this allows to claim that in average, there is no security loss to be expected if $(\delta_c + \delta_\sigma) \leq 2^{-46}$.

To check if this is the case for FALCON, we have run FALCON in two different precisions, a high precision of 200 bits and a standard precision of 53 bits, and compared the values of the c_j, σ_j's. The result of these experiments is that we always have $(\delta_c + \delta_\sigma) \leq 2^{-40}$: while this is higher than 2^{-46}, the difference is of only 6 bits. Therefore, we consider that 53 bits of precision are sufficient

for NIST's parameters (security level $\lambda \leq 256$, number of queries $q_s \leq 2^{64}$), and that the possibility of our signature procedure leaking information about the secret basis is a purely theoretic threat.

3.1.4. Advantages and Limitations of Falcon

This section lists the advantages and limitations of Falcon.

3.1.4.1. Advantages

Compactness. The main advantage of Falcon is its compactness. This doesn't really come as a surprise as Falcon was designed with compactness as the main criterion. Stateless hash-based signatures often have small public keys, but large signatures [BHH+15; AE17]. Conversely, some multivariate [KPG99; DS05] and code-based [CFS01] signature schemes achieve very small signatures but they require large public keys. Lattice-based schemes [DLL+17] can somewhat offer the best of both worlds, but we do not know of any post-quantum signature schemes getting $|\mathsf{pk}| + |\mathsf{sig}|$ to be as small as Falcon does.

Fast signature generation and verification. The signature generation and verification procedures are very fast. This is especially true for the verification algorithm, but even the signature algorithm can perform more than 1000 signatures per second on a moderately-powered computer.

Security in the ROM and QROM. The GPV framework comes with a security proof in the random oracle, and a security proof in the quantum random oracle model was later provided in [BDF+11]. This stands in contrast with schemes using the Fiat-Shamir heuristic, which are notoriously harder to render secure in the QROM [KLS17; Unr17].

Modular design. The design of Falcon is modular. Indeed, we instantiate the GPV framework with NTRU lattices, but it would be easy to replace NTRU lattices with another class of lattices if necessary. Similarly, we use fast Fourier sampling as our trapdoor sampler, but it is not necessary either. Actually, an extreme simplicity/speed trade-off would be to replace our fast Fourier sampler with Klein's sampler: signature generation would be about two orders of magnitudes slower, but it would be simpler to implement and the security would remain exactly the same.

Message recovery mode. In some situations, it can be advantageous to use Falcon in message-recovery mode. The signature becomes twice as long but the message does not need to be sent anymore, which induces a gain on the total communication complexity.

Identity-based encryption. As shown in [DLP14], Falcon can be converted into an identity-based encryption scheme in a straightforward manner.

Easy signature verification. The signature procedure is very simple: essentially, one just needs to compute $[H(\mathsf{r}\|\mathsf{m}) - s_2 h] \bmod q$, which boils down to a few NTT operations and a hash computation.

3.1.4.2. Limitations

Falcon also has a few limitations. These limitations are implementation-related and interestingly, they concern only the signer. We list them below.

Delicate implementation. We believe that both the key generation procedure and the fast Fourier sampling are non-trivial to understand and delicate to implement, and constitue the main shortcoming of Falcon. On the bright side, the fast Fourier sampling uses subroutines of the fast Fourier transform as well as trees, two objects most implementers are familiar with.

Floating-point arithmetic. Our signing procedure uses floating-point arithmetic with 53 bits of precision. While this poses no problem for a software implementation, it may prove to be a major limitation when implementation on constrained devices – in particular those without a floating-point unit – will be considered.

Unclear side-channel resistance. Falcon relies heavily on discrete Gaussian sampling over the integers. How to implement this securely with respect to timing and side-channel attacks has remained largely unstudied, save for a few exceptions [MW17; RRVV14].

3.2. The Falcon Signature Scheme

Main elements in Falcon are polynomials of degree n with integer coefficients. The degree n is normally a power of two (typically 512 or 1024) or a small multiple of a power of two (e.g. 768). Computations are done modulo a monic polynomial of degree n denoted ϕ (in practice, ϕ will be a cyclotomic polynomial).

Mathematically, within the algorithm, some polynomials are intepreted as vectors, and some others as matrices: a polynomial f modulo ϕ then stands for a square $n \times n$ matrix, whose rows are $x^i f \bmod \phi$ for all i from 0 to $n-1$. It can be shown that additions and multiplications of such matrices map to additions and multiplications of polynomials modulo ϕ. We can therefore express most of Falcon in terms of operations on polynomials, even when we really are handling matrices that define a *lattice*.

The public key is a basis for a lattice of dimension $2n$:

$$\left[\begin{array}{c|c} -h & I_n \\ \hline qI_n & O_n \end{array}\right] \tag{3.4}$$

where I_n is the identity matrix of dimension n, O_n contains only zeros, and h is a polynomial modulo ϕ that stands for an $n \times n$ sub-matrix, as explained above. Coefficients of h are integers that range from 0 to $q-1$, where q is a specific small prime (in the recommended parameters, q is either 12289 or 18433).

The corresponding private key is another basis for the very same lattice, expressed as:

$$\left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right] \tag{3.5}$$

where f, g, F and G are short integral polynomials modulo ϕ, that fulfill the two following relations:

$$\begin{array}{rcll} h & = & g/f & \bmod \phi \bmod q \\ fG - gF & = & q & \bmod \phi \end{array} \tag{3.6}$$

Such a lattice is known as a *complete NTRU lattice*, and the second relation, in particular, is called the *NTRU equation*. Take care that while the relation $h = g/f$ is expressed modulo q, the lattice itself, and the polynomials, use nominally unbounded integers.

Key pair generation involves choosing random f and g polynomials using an appropriate distribution that yields short, but not too short, vectors; then, the NTRU equation is solved to find matching F and G. Key and their generation are described in section 3.2.2.

Signature generation consists in first hashing the message to sign, along with a random nonce, into a polynomial c modulo ϕ, whose coefficients are uniformly mapped to integers in the 0 to $q-1$ range; this process is described in section 3.2.5. Then, the signer uses his knowledge of the secret lattice basis (f, g, F, G) to produce a pair of short polynomials (s_1, s_2) such that $s_1 = c - s_2 h \bmod \phi \bmod q$. The signature properly said is s_2.

Finding small vectors s_1 and s_2 is, in all generality, an expensive process. Falcon leverages the special structure of ϕ to implement it as a divide-and-conquer algorithm similar to the Fast Fourier Transform, which greatly speeds up operations. Moreover, some "noise" is added to the sampled vectors, with carefully tuned Gaussian distributions, to prevent signatures from leaking too much information about the private key. The signature generation process is described in section 3.2.6.

Signature verification consists in recomputing s_1 from the hashed message c and the signature s_2, and then verifying that (s_1, s_2) is an appropriately short vector. Signature verification can be done entirely with integer computations modulo q; it is described in section 3.2.7.

Encoding formats for keys and signatures are described in section 3.2.8. In particular, since the signature is a short polynomial s_2, its elements are on average close to 0, which allows for a custom compressed format that reduces signature size.

Recommended parameters for several security levels are defined in section 3.2.9.

3.2.1. Preliminaries

In this section, we provide an overview of the used techniques. As Falcon is arguably math-heavy, a clear comprehension of the mathematical principles in action goes a long way towards understanding and implementing it.

Falcon works with elements in number fields of the form $\mathbb{Q}[x]/(\phi)$. Here ϕ denotes a cyclotomic polynomial, that is, a polynomial of the form $\phi(x) = \prod_{\zeta \in \Omega}(x - \zeta)$ where Ω denotes the set of primitive m-th roots of unity for an integer m. In particular, we will always use one of these two types of polynomials:

- $\phi = x^n + 1$ for $n = 2^\kappa$; as this polynomial is binary and will entail manipulating binary trees, we will say that we are in the *binary case* whenever we work with it; we note that in this case $\phi(x) = \prod_{k \in \mathbb{Z}_m^\times}(x - \zeta^k)$, with $m = 2n$ and ζ an arbitrary primitive m-th root of 1 (e.g. $\zeta = \exp(\frac{2i\pi}{m})$).
- $\phi = x^n - x^{n/2} + 1$ with $n = 3 \cdot 2^\kappa$; as this polynomial is ternary and will entail manipulating ternary trees, we will say that we are in the *ternary case* whenever we work with it. We note that in this case $\phi(x) = \prod_{k \in \mathbb{Z}_m^\times}(x - \zeta^k)$, where $m = 3n$ and ζ is a primitive m-th root of 1.

The interesting part about these number fields $\mathbb{Q}[x]/(\phi)$ is that they come with a tower-of-fields structure. Indeed, in the binary case, we have the following tower of fields:

$$\mathbb{Q} \subseteq \mathbb{Q}[x]/(x^2 + 1) \subseteq \cdots \subseteq \mathbb{Q}[x]/(x^{n/2} + 1) \subseteq \mathbb{Q}[x]/(x^n + 1) \tag{3.7}$$

Similarly, in the ternary case, we have the following tower of fields:

$$\begin{array}{rcl} \mathbb{Q} & \subseteq & \mathbb{Q}[x]/(x^2 - x + 1) \subseteq \mathbb{Q}[x]/(x^6 - x^3 + 1) \subseteq \mathbb{Q}[x]/(x^{12} - x^6 + 1) \subseteq \ldots \\ \ldots & \subseteq & \mathbb{Q}[x]/(x^{n/2} - x^{n/4} + 1) \subseteq \mathbb{Q}[x]/(x^n - x^{n/2} + 1) \end{array} \tag{3.8}$$

In both the binary and ternary cases, we will rely on this tower-of-fields structure. Even more importantly for our purposes, by splitting polynomials between their odd and even coefficients we have the following chain of space isomorphisms:

$$\mathbb{Q}^n \cong (\mathbb{Q}[x]/(x^2 + 1))^{n/2} \cong \ldots \cong (\mathbb{Q}[x]/(x^{n/2} + 1))^2 \cong \mathbb{Q}[x]/(x^n + 1) \tag{3.9}$$

Similarly, in the ternary case, we have the following chain of space isomorphisms:

$$\mathbb{Q}^n \cong (\mathbb{Q}[x]/(x^2 - x + 1))^{n/2} \cong (\mathbb{Q}[x]/(x^6 - x^3 + 1))^{n/6} \cong \ldots \cong \mathbb{Q}[x]/(x^n - x^{n/2} + 1) \tag{3.10}$$

The equations 3.7, 3.8, 3.9 and 3.10 remain valid when replacing $\mathbb{Q}$ by $\mathbb{Z}$, in which case they describe a tower of rings and a chain of module isomorphisms.

We will see in section 3.2.4 that for appropriately defined multiplications, these are actually chains of *ring* isomorphisms. The equations 3.9 and 3.10 will be used to make our signature generation fast and "good": in lattice-based cryptography, the smaller the norm of signatures are, the better. So by "good" we mean that our signature generation will output signatures with a small norm.

On one hand, classical algebraic operations in the fields $\mathbb{Q}[x]/(x^n - x^{n/2} + 1)$ and $\mathbb{Q}[x]/(x^n + 1)$ are fast, and using them will make our signature generation fast. On the other hand, we will use the isomorphisms exposed in equations 3.9 and 3.10 as a leverage to output signatures with small norm. However, using these endomorphisms to their full potential is not easy, as it entails manipulating individual coefficients of polynomials (or of their Fourier transform) and working with binary or even ternary trees. We will see that most of the technicalities of Falcon arise from this.

Cryptographic parameters. For a cryptographic signature scheme, λ denotes its security level and q_s the maximal number of signature queries which may be made. Following the assumptions of [NIS16], we suppose that $q_s \leq 2^{64}$.

Matrices, vectors and scalars. Matrices will usually be in bold uppercase (e.g. $\mathbf{B}$), vectors in bold lowercase (e.g. $\mathbf{v}$) and scalars – which include polynomials – in italic (e.g. s). We use the row convention for vectors. The transpose of a matrix $\mathbf{B}$ may be noted $\mathbf{B}^t$. It is to be noted that for a polynomial f, we do *not* use f' to denote its derivative in this document.

Quotient rings. For $q \in \mathbb{N}^\star$, we denote by $\mathbb{Z}_q$ the quotient ring $\mathbb{Z}/q\mathbb{Z}$; in Falcon, q is prime so $\mathbb{Z}_q$ becomes a finite field. We also denote by $\mathbb{Z}_q^\times$ the group of invertible elements of $\mathbb{Z}_q$, and by φ Euler's totient function: $\varphi(q) = |\mathbb{Z}_q^\times|$.

Number fields. We denote by ϕ a monic polynomial of $\mathbb{Z}[x]$, irreducible in $\mathbb{Q}[x]$, of degree n and with distinct roots over $\mathbb{C}$. In Falcon, we will always consider ϕ to take one of these two forms:

- *Binary case.* $\phi = x^n + 1$ for $n = 2^\kappa$;
- *Ternary case.* $\phi = x^n - x^{n/2} + 1$ with $n = 3 \cdot 2^\kappa$.

Let $a = \sum_{i=0}^{n-1} a_i x^i$ and $b = \sum_{i=0}^{n-1} b_i x^i$ be arbitrary elements of the number field $\mathcal{Q} = \mathbb{Q}[x]/(\phi)$. We note $a^\star$ and call (Hermitian) adjoint of a the unique element of $\mathcal{Q}$ such that for any root ζ of ϕ, $a^\star(\zeta) = \overline{a(\zeta)}$, where $\bar{\cdot}$ is the usual complex conjugation over $\mathbb{C}$. For the values of ϕ considered in Falcon, $a^\star$ can be expressed simply:

- *Binary case.* If $\phi = x^n + 1$ with $n = 2^\kappa$ a power of 2, then

$$a^\star = a_0 - \sum_{i=1}^{n-1} a_i x^{n-i} \tag{3.11}$$

- *Ternary case.* If $\phi = x^n - x^{n/2} + 1$ with $n = 3 \cdot 2^\kappa$, then

$$a^\star = a_0 + \sum_{i=1}^{n-1} a_i (x^{n/2-i} - x^{n-i}) \tag{3.12}$$

We extend this definition to vectors and matrices: the adjoint $\mathbf{B}^\star$of a matrix $\mathbf{B} \in \mathcal{Q}^{n\times m}$ (resp. a vector $\mathbf{v}$) is the component-wise adjoint of the transpose of $\mathbf{B}$ (resp. $\mathbf{v}$).

The inner product over $\mathcal{Q}$ is $\langle a, b\rangle = \frac{1}{\deg(\phi)} \sum_{\phi(\zeta)=0} a(\zeta) \cdot \overline{b(\zeta)}$, and the associated norm is $\|a\| = \sqrt{\langle a, a\rangle}$. We extend this definition to vectors: for $\mathbf{u} = (u_i)_i$ and $\mathbf{v} = (v_i)_i$ in $\mathcal{Q}^m$, we define $\langle \mathbf{u}, \mathbf{v}\rangle$ as $\sum_i \langle u_i, v_i\rangle$. Of special interest to us is the expression of the norm for specific values of ϕ.

- *Binary case.* The norm coincides with the usual coefficient-wise euclidean norm:

$$\|a\|^2 = \sum_{0\le i<n} a_i^2; \tag{3.13}$$

- *Ternary case.* The norm can be expressed as:

$$\|a\|^2 = \sum_{0\le i<n/2} (a_i^2 + a_i a_{i+n/2} + a_{i+n/2}^2). \tag{3.14}$$

Ring Lattices. For the rings $\mathcal{Q} = \mathbb{Q}[x]/(\phi)$ and $\mathcal{Z} = \mathbb{Z}[x]/(\phi)$, positive integers $m \ge n$ and a full-rank matrix $\mathbf{B} \in \mathcal{Q}^{n\times m}$, we denote by $\Lambda(\mathbf{B})$ and call lattice generated by $\mathbf{B}$ the set $\mathcal{Z}^n \cdot \mathbf{B} = \{\mathbf{zB} | \mathbf{z} \in \mathcal{Z}^n\}$. By extension, a set Λ is a lattice if there exists a matrix $\mathbf{B}$ such that $\Lambda = \Lambda(\mathbf{B})$. We may say that $\Lambda \subseteq \mathcal{Z}^m$ is a q-ary lattice if $q\mathcal{Z}^m \subseteq \Lambda$.

Discrete Gaussians. For $\sigma, \mu \in \mathbb{R}$ with $\sigma > 0$, we define the Gaussian function $\rho_{\sigma,\mu}$ as $\rho_{\sigma,\mu}(x) = \exp(-|x-\mu|^2/2\sigma^2)$, and the discrete Gaussian distribution $D_{\mathbb{Z},\sigma,\mu}$ over the integers as

$$D_{\mathbb{Z},\sigma,\mu}(x) = \frac{\rho_{\sigma,\mu}(x)}{\sum_{z\in\mathbb{Z}} \rho_{\sigma,\mu}(z)}. \tag{3.15}$$

The parameter μ may be omitted when it is equal to zero.

Field norm. Let $\mathbb{K}$ be a number field of degree $n = [\mathbb{K} : \mathbb{Q}]$ over $\mathbb{Q}$ and $\mathbb{L}$ be a Galois extension of $\mathbb{K}$. We denote by $\mathrm{Gal}(\mathbb{L}/\mathbb{K})$ the Galois group of $\mathbb{L}/\mathbb{K}$. The field norm $\mathrm{N}_{\mathbb{L}/\mathbb{K}} : \mathbb{L} \to \mathbb{K}$ is a map defined for any $f \in \mathbb{L}$ by the product of the Galois conjugates of f:

$$\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \prod_{\mathrm{g}\in\mathrm{Gal}(\mathbb{L}/\mathbb{K})} \mathrm{g}(f). \tag{3.16}$$

Equivalently, $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f)$ can be defined as the determinant of the $\mathbb{K}$-linear map $y \in \mathbb{L} \mapsto fy$. One can check that the field norm is a multiplicative morphism.

The Gram-Schmidt orthogonalization. Any matrix $\mathbf{B} \in \mathcal{Q}^{n \times m}$ can be decomposed as follows:

$$\mathbf{B} = \mathbf{L} \times \tilde{\mathbf{B}}, \tag{3.17}$$

where $\mathbf{L}$ is lower triangular with 1's on the diagonal, and the rows $\tilde{\mathbf{b}}_i$'s of $\tilde{\mathbf{B}}$ verify $\mathbf{b}_i \mathbf{b}_j^\star = 0$ for $i \neq j$. When $\mathbf{B}$ is full-rank , this decomposition is unique, and it is called the Gram-Schmidt orthogonalization (or GSO). We will also call Gram-Schmidt norm of $\mathbf{B}$ the following value:

$$\|\mathbf{B}\|_{\text{GS}} = \max_{\tilde{\mathbf{b}}_i \in \tilde{\mathbf{B}}} \|\tilde{\mathbf{b}}_i\|. \tag{3.18}$$

The LDL* decomposition. Closely related to the GSO is the LDL* decomposition. It writes any full-rank Gram matrix as a product $\mathbf{LDL}^\star$, where $\mathbf{L} \in \mathcal{Q}^{n \times n}$ is lower triangular with 1's on the diagonal, and $\mathbf{D} \in \mathcal{Q}^{n \times n}$ is diagonal. It can be computed using algorithm 3.26.

The LDL* decomposition and the GSO are closely related as for a basis $\mathbf{B}$, there exists a unique GSO $\mathbf{B} = \mathbf{L} \cdot \tilde{\mathbf{B}}$ and for a full-rank Gram matrix $\mathbf{G}$, there exists a unique LDL* decomposition $\mathbf{G} = \mathbf{LDL}^\star$. If $\mathbf{G} = \mathbf{BB}^\star$, then $\mathbf{G} = \mathbf{L} \cdot (\tilde{\mathbf{B}}\tilde{\mathbf{B}}^\star) \cdot \mathbf{L}^\star$ is a valid LDL* decomposition of $\mathbf{G}$. As both decompositions are unique, the matrices $\mathbf{L}$ in both cases are actually the same. In a nutshell:

$$\left[\mathbf{L} \cdot \tilde{\mathbf{B}} \text{ is the GSO of } \mathbf{B}\right] \Leftrightarrow \left[\mathbf{L} \cdot (\tilde{\mathbf{B}}\tilde{\mathbf{B}}^\star) \cdot \mathbf{L}^\star \text{ is the LDL}^\star \text{ decomposition of } (\mathbf{BB}^\star)\right]. \tag{3.19}$$

The reason why we present both equivalent decompositions is because the GSO is a more familiar concept in lattice-based cryptography, whereas the use of LDL* decomposition is faster and therefore makes more sense from an algorithmic point of view.

3.2.2. Key Pair Generation

3.2.2.1. Public Parameters

Public keys use some public parameters that are shared by many key pairs:

1. A cyclotomic polynomial $\phi \in \mathbb{Z}[x]$, which is monic and irreducible. In Falcon, ϕ is either of these two types of polynomials:
   - *Binary case.* $\phi = x^n + 1$, where $n = 2^\kappa$ is a power of 2;
   - *Ternary case.* $\phi = x^n - x^{n/2} + 1$, where $n = 3 \cdot 2^\kappa$ is 3 times a power of 2;
2. A modulus $q \in \mathbb{N}^\star$. In Falcon, q may take either of these two values:
   - *Binary case.* If $n = 2^\kappa$ is a power of 2, then $q = 12289$;
   - *Ternary case.* If $n = 3 \cdot 2^\kappa$ is 3 times a power of 2, then $q = 18433$;

   In both cases, q is chosen so that $(\phi \bmod q)$ splits over $\mathbb{Z}_q[x]$.
3. A real bound $\beta > 0$.

For clarity, all the parameters presented may be omitted (e.g. in algorithms' headers) when clear from context.

3.2.2.2. Private Key

The core of a FALCON private key sk consists of four polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ with short integer coefficients, verifying the NTRU equation:

$$fG - gF = q \bmod \phi. \tag{3.20}$$

The polynomial f shall furthermore be invertible in $\mathbb{Z}_q[x]/(\phi)$.

Given f and g such that there exists a solution (F, G) to the NTRU equation, F and G may be recomputed dynamically, but that process is computationally expensive; therefore, it is normally expected that at least F will be stored along f and g (given f, g and F, G can be efficiently recomputed).

Two additional elements are computed from the private key, and may be recomputed dynamically, or stored along f, g and F:

- The FFT representations of f, g, F and G, ordered in the form of a matrix:

$$\hat{\mathbf{B}} = \left[\begin{array}{c|c} \mathsf{FFT}(g) & -\mathsf{FFT}(f) \\ \hline \mathsf{FFT}(G) & -\mathsf{FFT}(F) \end{array}\right], \tag{3.21}$$

  where $\mathsf{FFT}(a)$ denotes the fast Fourier transform of a in the underlying ring (here, the ring is $\mathbb{R}[x]/(\phi)$).

- A FALCON tree T, described at the end of this section. An important subtlety in FALCON is that the nature of the FALCON tree differs depending on ϕ: in the binary case (i.e. $\phi = x^n + 1$), the FALCON tree is a binary tree – each node has at most two children –, and in the ternary case (i.e. $\phi = x^n - x^{n/2} + 1$), the FALCON tree is a ternary tree – each node has at most three children.

FFT representations are described in section 3.2.3. The FFT representation of a polynomial formally consists of n complex numbers (a complex number is normally encoded as two 64-bit floating-point values); however, the FFT representation of a *real* polynomial f is redundant, because for each complex root ζ of ϕ, its conjugate $\overline{\zeta}$ is also a root of ϕ, and $f(\overline{\zeta}) = \overline{f(\zeta)}$. Therefore, the FFT representation of a polynomial may be stored as $n/2$ complex numbers, and $\hat{\mathbf{B}}$, when stored, requires $2n$ complex numbers.

FALCON binary trees. FALCON binary trees are defined inductively as follows:

- A FALCON binary tree T of height 0 consists of a single node whose value is a real $\sigma > 0$.
- A FALCON binary tree T of height κ verifies these properties:
  - The value of its root, noted T.value, is a polynomial $\ell \in \mathbb{Q}[x]/(x^n + 1)$ with $n = 2^\kappa$.
  - Its left and right children, noted T.leftchild and T.rightchild, are FALCON binary trees of height $\kappa - 1$.

The values of internal nodes – which are real polynomials – are stored in FFT representation (i.e. as complex numbers, see section 3.2.3 for a formal definition). Hence all the nodes of a FALCON binary tree contain polynomials in FFT representation, except the leaves which contain real values > 0. A FALCON binary tree of height 3 is represented in the figure 3.2. As illustrated by the figure, a FALCON binary tree can be easily represented by an array of $2^\kappa(1 + \kappa)$ complex numbers (or exactly half as many, if the redundancy of FFT representation is leveraged, as explained above), and access to the left and right children can be performed efficiently using simple pointer arithmetic.

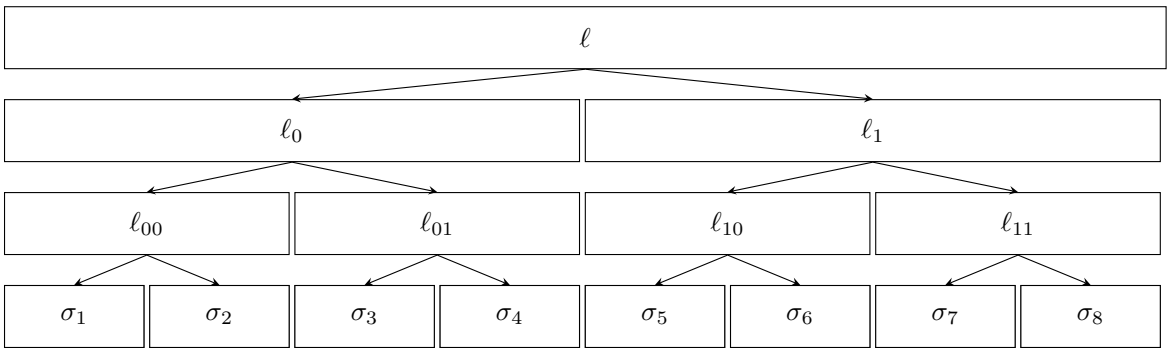


Figure 3.2. – A Falcon binary tree of height 3

Falcon ternary trees. In the ternary case, a mostly binary tree is built, except in one level whose nodes have three children instead of two. Generally speaking, leaf nodes correspond to degree 1, and each upper level either doubles or triples the degree; the tree root then corresponds to the degree n of ϕ. Since $n = 3 \cdot 2^\kappa$, there must be exactly one level that corresponds to a "degree tripling"; nodes in that level have three children each. Exactly which level consists in ternary nodes is in fact left open as an implementation choice. The description of Falcon in this specification corresponds to a tripling immediately above the leaf nodes. As in the binary case, the index of each node in a continuous array representation can be efficiently computed. The contents of a Falcon tree T are computed from the private key elements f, g, F and G using the algorithms described in section 3.2.2.6.

3.2.2.3. Public key

The Falcon public key pk corresponding to the private key sk $= (f, g, F, G)$ is a polynomial $h \in \mathbb{Z}_q[x]/(\phi)$ such that:

$$h = gf^{-1} \bmod (\phi, q). \tag{3.22}$$

3.2.2.4. Key Pair Generation

The key pair generation is arguably the most technical part of Falcon to describe. It can be chronologically and conceptually decomposed in two clearly separate parts, which each contain their own technicalities, make use of different mathematical tools and require to address different challenges.

- *Solving the NTRU equation.* The first step of the key pair generation consists of computing polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ which verify the equation 3.20 – the NTRU equation. Generating f and g is easy enough, but the hard part is to compute efficiently polynomials F, G such that equation 3.20 is verified.

  In order to do this, we propose a novel method which exploits the tower-of-rings structure explicited in the equations 3.7 and 3.8. We use the field norm N to map the NTRU equation onto a smaller ring $\mathbb{Z}[x]/(\phi')$ of the tower of rings, all the way down to $\mathbb{Z}$. We then solve the equation in $\mathbb{Z}$ – which amounts to an extended gcd – and use the properties of the norm to lift the solutions (F, G) in the tower of rings, up to the ring $\mathbb{Z}[x]/(\phi)$.

From an implementation point of view, the main technicality of this part is that it requires to handle polynomials with large coefficients (a few thousands of bit per coefficient in the lowest levels of the recursion). This step is specified in section 3.2.2.5.

- *Computing a Falcon tree.* Once suitable polynomials f, g, F, G are generated, the second part of the key generation consists of preprocessing them into an adequate format: by adequate we mean that this format should be reasonably compact and allow fast signature generation on-the-go.

  Falcon trees are precisely this adequate format. To compute a Falcon tree, we compute the LDL* decomposition $\mathbf{G} = \mathbf{LDL}^\star$ of the matrix $\mathbf{G} = \mathbf{BB}^\star$, where

  $$\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right], \tag{3.23}$$

  which is equivalent to computing the Gram-Schmidt orthogonalization $\mathbf{B} = \mathbf{L} \times \tilde{\mathbf{B}}$. If we were using Klein's well-known sampler (or a variant thereof) as a trapdoor sampler, knowing $\mathbf{L}$ would be sufficient but a bit unsatisfactory as we would not exploit the tower-of-rings structure of $\mathbb{Q}[x]/(\phi)$.

  So instead of stopping there, we store $\mathbf{L}$ (or rather L_{10}, its only non-trivial term) in the root of a tree, use the splitting operators defined in section 3.2.4 to "break" the diagonal elements D_{ii} of $\mathbf{D}$ into matrices $\mathbf{G}_i$ over smaller rings $\mathbb{Q}[x]/(\phi')$, at which point we create subtrees for each matrix $\mathbf{G}_i$ and recursively start over the process of LDL* decomposition and splitting.

  The recursion continues until the matrix $\mathbf{G}$ has its coefficients in $\mathbb{Q}$, which correspond to the bottom of the recursion tree. How this is done is specified in section 3.2.2.6.

  The main technicality of this part is that it exploits the tower-of-rings structure of $\mathbb{Q}[x]/(\phi)$ by breaking its elements onto smaller rings. In addition, intermediate results are stored in a tree, which requires precise bookkeeping as elements of different tree levels do not live in the same field. Finally, for performance reasons, the step is realized completely in the FFT domain.

Once these two steps are done, the rest of the key pair generation is straightforward. A final step normalizes the leaves of the LDL tree to turn it into a Falcon tree. The result is wrapped in a private key sk and the corresponding public key pk is $h = gf^{-1} \bmod q$.

A formal description is given in the algorithms 3.20 to 3.28, the main algorithm being the procedure Keygen (algorithm 3.20). The general architecture of the key pair generation is also illustrated in figure 3.3.

3.2.2.5. Generating the polynomials f, g, F, G.

The first step of the key pair generation generates suitable polynomials f, g, F, G verifying the NTRU equation: $fG - gF = q \bmod \phi$. This is specified in algorithm 3.21 (NTRUGen). We provide a general explanation of the algorithm:

1. At the first step, the polynomials f, g are generated randomly. A few conditions over f, g are checked to ensure they are suitable for our purposes (steps 1 to 11). It particular, it shall be checked that:

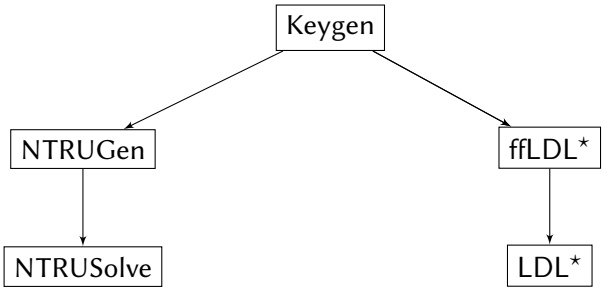


Figure 3.3. – Flowchart of the key generation

Algorithm 3.20 Keygen(ϕ, q)

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$, a modulus q
Ensure: A secret key sk, a public key pk

$f, g, F, G, \gamma \leftarrow \mathsf{NTRUGen}(\phi, q)$ ▷ Solving the NTRU equation
$\mathbf{B} \leftarrow \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$
$\hat{\mathbf{B}} \leftarrow \mathsf{FFT}(\mathbf{B})$
$\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^{\star}$
$\mathsf{T} \leftarrow \mathsf{ffLDL}^{\star}(\mathbf{G})$ ▷ Computing the LDL* tree
if ϕ is binary **then**
    $\sigma \leftarrow 1.55\sqrt{q}$
else if ϕ is ternary **then**
    $\sigma \leftarrow 1.32 \cdot 2^{1/4}\sqrt{q}$
for each leaf leaf of T **do** ▷ Normalization step
    $\mathsf{leaf.value} \leftarrow \sigma/\sqrt{\mathsf{leaf.value}}$
$\mathsf{sk} \leftarrow (\hat{\mathbf{B}}, \mathsf{T})$
$h \leftarrow gf^{-1} \bmod q$
$\mathsf{pk} \leftarrow h$
return sk, pk

a) A public key h can be computed from f, g. This is true if and only if f is invertible $\bmod q$, which is true if and only if $\mathrm{Res}(f, \phi) \bmod q \neq 0$, where Res denotes the resultant. The NTT can be used to perform this check and then compute h (f is invertible in $\mathbb{Z}_q[x]/(\phi)$ if and only if its NTT representation contains no zero).

b) The polynomials f, g, F, G allow to generate short signatures. This is the case if and only if $\gamma = \max\left\{\|(g, -f)\|, \left\|\left(\frac{qf^\star}{ff^\star+gg^\star}, \frac{qg^\star}{ff^\star+gg^\star}\right)\right\|\right\}$ is small enough.

2. At the second step, short polynomials F, G are computed such that f, g, F, G verify the NTRU equation. This is done by the procedure NTRUSolve, which exists in two versions: binary (algorithm 3.23) and ternary (algorithm 3.25).

Algorithm 3.21 NTRUGen(ϕ, q) (Binary case)

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$ of degree n, a modulus q
Ensure: Polynomials f, g, F, G

$\sigma' \leftarrow 1.17\sqrt{q/2n}$ ▷ σ' is chosen so that $\mathbb{E}[\|(f, g)\|] = 1.17\sqrt{q}$
for i from 0 to $n-1$ **do**
  $f_i \leftarrow D_{\mathbb{Z},\sigma',0}$
  $g_i \leftarrow D_{\mathbb{Z},\sigma',0}$
$f \leftarrow \sum_i f_i x^i$ ▷ $f \in \mathbb{Z}[x]/(\phi)$
$g \leftarrow \sum_i g_i x^i$ ▷ $g \in \mathbb{Z}[x]/(\phi)$
if $\mathrm{Res}(f, \phi) \bmod q = 0$ **then** ▷ Check that f is invertible $\bmod q$
  restart
$\gamma \leftarrow \max\left\{\|(g, -f)\|, \left\|\left(\frac{qf^\star}{ff^\star+gg^\star}, \frac{qg^\star}{ff^\star+gg^\star}\right)\right\|\right\}$
if $\gamma > 1.17\sqrt{q}$ **then** ▷ Check that signatures will be short
  restart
$F, G \leftarrow \mathsf{NTRUSolve}_{n,q}(f, g)$ ▷ Computing F, G such that $fG - gF = 1 \bmod \phi$
return f, g, F, G

Things are slightly different in the ternary case: to ensure the proper distribution, coefficients of f and g must be generated as Gaussians in the FFT embedding, then converted back to normal representation, and only then rounded to integers. Moreover, the vector norms must be evaluated in the FFT embedding as well. This is due to the fact that the FFT, in the ternary case, is not an orthogonal transform, and we need f and g to use a proper spheroid in the FFT embedding. Details are expressed in algorithm 3.22.

Solving the NTRU equation: the binary case. We now explain how to solve the equation 3.20 in the binary case. As said before, we repeatedly use the field norm N (see section 3.2.4.5 for explicit formulae) to map f, g to a smaller ring $\mathbb{Z}[x]/(x^{n/2}+1)$, until we reach the ring $\mathbb{Z}$. Solving 3.20 then amounts to computing an extended GCD over $\mathbb{Z}$, which is simple. Once this is done, we use the multiplicative properties of the field norm to repeatedly lift the solutions up to $\mathbb{Z}[x]/(x^n+1)$, at which point we have solved the equation 3.20. NTRUSolve uses the procedure Reduce as a subroutine to reduce the size of the solutions F, G. Unlike NTRUSolve, the description of this subroutine is the same in the binary and ternary cases. The principle of Reduce is a simple generalization of textbook vectors' reduction. Given vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^k$, reducing $\mathbf{u}$ with respect to $\mathbf{v}$ is done by simply performing $\mathbf{u} \leftarrow \mathbf{u} - \left\lfloor \frac{\mathbf{uv}^\star}{\mathbf{vv}^\star} \right\rceil \mathbf{v}$. Reduce does the same by replacing $\mathbb{Z}^k$ by $(\mathbb{Z}[x]/(\phi))^2$, $\mathbf{u}$ by (F, G) and $\mathbf{v}$ by (f, g).

Algorithm 3.22 NTRUGen(ϕ, q) (Ternary case)

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$ of degree n, a modulus q
Ensure: Polynomials f, g, F, G

$\sigma' \leftarrow \sqrt{q/\sqrt{8}}$
for j from 0 to $n-1$ **do**
  Generate random uniform real numbers $a, b, c, d \in]0..1]$
  $\hat{f}_j \leftarrow \sigma' \sqrt{-2\log a} e^{i(2\pi)b}$
  $\hat{g}_j \leftarrow \sigma' \sqrt{-2\log c} e^{i(2\pi)d}$
$f \leftarrow \mathsf{invFFT}(\hat{f})$
$g \leftarrow \mathsf{invFFT}(\hat{g})$
for j from 0 to $n-1$ **do**
  $f_j \leftarrow \lfloor f_j \rceil$
  $g_j \leftarrow \lfloor g_j \rceil$
if $\mathrm{Res}(f, \phi) \bmod q = 0$ **then** ▷ Check that f is invertible $\bmod q$
  restart
$\gamma \leftarrow \max\left\{ \|(g, -f)\|, \left\| \left(\frac{qf^\star}{ff^\star + gg^\star}, \frac{qg^\star}{ff^\star + gg^\star}\right) \right\| \right\}$
if $\gamma > 4nq/\sqrt{8}$ **then** ▷ Check that signatures will be short
  restart
$F, G \leftarrow \mathsf{NTRUSolve}_{n,q}(f, g)$ ▷ Computing F, G such that $fG - gF = 1 \bmod \phi$
return f, g, F, G

Algorithm 3.23 $\mathsf{NTRUSolve}_{n,q}(f, g)$ (Binary case)

Require: $f, g \in \mathbb{Z}[x]/(x^n + 1)$ with n a power of two
Ensure: Polynomials F, G such that the equation 3.20 is verified

if $n = 1$ **then**
  Compute $u, v \in \mathbb{Z}$ such that $uf - vg = \gcd(f, g)$
  if $\gcd(f, g) \neq 1$ **then**
    abort
  $(F, G) \leftarrow (vq, uq)$
  return (F, G)
else
  $f' \leftarrow \mathrm{N}(f)$ ▷ $f', g', F', G' \in \mathbb{Z}[x]/(x^{n/2} + 1)$
  $g' \leftarrow \mathrm{N}(g)$
  $(F', G') \leftarrow \mathsf{NTRUSolve}_{n/2,q}(f', g')$
  $F \leftarrow F'(x^2)g'(x^2)/g(x)$ ▷ $F, G \in \mathbb{Z}[x]/(x^n + 1)$
  $G \leftarrow G'(x^2)g'(x^2)/g(x)$
  $\mathsf{Reduce}(f, g, F, G)$ ▷ (F, G) is reduced with respect to (f, g)
return (F, G)

Algorithm 3.24 Reduce(f, g, F, G)

Require: Polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$
Ensure: (F, G) is reduced with respect to (f, g)

1: **do**
2: $k \leftarrow \left\lfloor \frac{Ff^\star + Gg^\star}{ff^\star + gg^\star} \right\rceil$ ▷ $\frac{Ff^\star+Gg^\star}{ff^\star+gg^\star} \in \mathbb{Q}[x]/(\phi)$ and $k \in \mathbb{Z}[x]/(\phi)$
3: $F \leftarrow F - kf$
4: $G \leftarrow G - kg$
5: **while** $k \neq 0$ ▷ Multiple iterations may be needed, e.g. if k is computed in small precision.

Solving the NTRU equation: the ternary case. For the ternary case, the principle of NTRUSolve is the same, except that the recursion is less straightforward as there are more cases to consider. However, the polynomials $\mathrm{N}(f)(x^k)/f(x)$ can still be expressed simply. Reprising the notations of section 3.2.4.5:

1. If $n > 6$, then $\mathrm{N}(f)(x^2)/f(x) = f(-x) = f_0(x^2) - xf_1(x^2)$.
2. If $n = 6$, then $\mathrm{N}(f)(x^3)/f(x) = f(x^7)f(x^{13})$. This is equal to
$$f_0^2(x^3) + x^2 f_1^2(x^3) + x^4 f_2^2(x^3) - xf_0f_1(x^3) - x^2 f_0 f_2(x^3) - x^3 f_1 f_2(x^3).$$
3. If $n = 2$, then $\mathrm{N}(f)(x^2)/f(x) = f(x^5) = f_0 + f_1 - xf_1$.

The ternary version of NTRUSolve is more complex than the binary; a complete specification is given in algorithm 3.25.

3.2.2.6. Computing a Falcon Tree

The second step of the key generation consists of preprocessing the polynomials f, g, F, G into an adequate secret key format. The secret key is of the form $\mathsf{sk} = (\hat{\mathbf{B}}, \mathsf{T})$, where:

- $\hat{\mathbf{B}} = \left[\begin{array}{c|c} \mathsf{FFT}(g) & -\mathsf{FFT}(f) \\ \hline \mathsf{FFT}(G) & -\mathsf{FFT}(F) \end{array}\right]$
- T is a Falcon tree computed in two steps:
  1. First, a tree T is computed from $\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^\star$, called an *LDL tree*. This is specified in algorithm 3.27. At this point, T is a Falcon tree but it is not normalized.
  2. Second, T is normalized with respect to a standard deviation σ. It is described near the end of algorithm 3.20.

  The polynomials manipulated in the algorithm 3.27 and its subroutine algorithm 3.26 are all in FFT representation. While it is possible to convert these algorithms to the coefficient representation, doing so would be suboptimal from an efficiency viewpoint.

At a high level, the method for computing the LDL tree at step 1 (before normalization) is simple:

1. We compute the LDL decomposition of $\mathbf{G}$: we write $\mathbf{G} = \mathbf{L} \times \mathbf{D} \times \mathbf{L}^\star$, with $\mathbf{L}$ a lower triangular matrix with 1's on the diagonal and $\mathbf{D}$ a diagonal matrix. Such a decomposition is easy to compute: we recall a method for computing the LDL decomposition in algorithm 3.26.

   We store the matrix $\mathbf{L}$ in T.value, which is the value of the root of T. Since $\mathbf{L}$ is a lower triangular matrix with 1's on the diagonal of dimensions – in our case – (2×2) or (3×3), this only amounts to storing one or three elements of $\mathbb{Q}[x]/(\phi)$.

Algorithm 3.25 $\mathsf{NTRUSolve}_{n,q}(f, g)$ (Ternary case)

Require: $f, g \in \mathbb{Z}[x]/(x^n - x^{n/2} + 1)$ with $n = 3 \cdot 2^\kappa$
Ensure: Polynomials F, G such that the equation 3.20 is verified

if $n = 1$ **then**
  Compute $u, v \in \mathbb{Z}$ such that $uf - vg = \gcd(f, g)$
  if $\gcd(f, g) \neq 1$ **then**
    abort
  $(F, G) \leftarrow (vq, uq)$
  return (F, G)
else
  if $n = 6$ **then**
    $k \leftarrow 3$
  else
    $k \leftarrow 2$
  $f' \leftarrow \mathrm{N}(f)$ ▷ $f', g', F', G' \in \mathbb{Z}[x]/(x^{n/k} - x^{n/(2k)} + 1)$
  $g' \leftarrow \mathrm{N}(g)$
  $(F', G') \leftarrow \mathsf{NTRUSolve}_{(n/k),q}(f', g')$
  $F \leftarrow F'(x^k) \cdot g'(x^k)/g(x)$ ▷ $F, G \in \mathbb{Z}[x]/(x^n - x^{n/2} + 1)$
  $G \leftarrow G'(x^k) \cdot f'(x^k)/f(x)$
  $\mathsf{Reduce}(f, g, F, G)$
return (F, G)

2. We then use the splitting operator to "break" each diagonal element of $\mathbf{D}$ into a matrix of smaller elements. More precisely, for a diagonal element $d \in \mathbb{Q}[x]/(\phi)$, we consider the associated endomorphism $\psi_d : z \in \mathbb{Q}[x]/(\phi) \mapsto dz$ and write its transformation matrix over the smaller ring $\mathbb{Q}[x]/(\phi')$.

   a) *Binary case.* If $\phi = x^n + 1$, then we take $\phi' = x^{n/2} + 1$. Following the argument of section 3.2.4.4, the transformation matrix of ψ_d can be written as

$$\left[\begin{array}{c|c} d_0 & d_1 \\ \hline xd_1 & d_0 \end{array}\right] \left(= \left[\begin{array}{c|c} d_0 & d_1 \\ \hline d_1^\star & d_0 \end{array}\right]\right)^{1}. \tag{3.24}$$

   b) *Ternary case.* If $\phi = x^n - x^{n/2} + 1$, then depending on the value of n we may take either $\phi' = x^{n/2} - x^{n/4} + 1$ or $\phi' = x^{n/3} - x^{n/6} + 1$. The first case amounts to splitting d in two, and we can then simply use the equation 3.24. In the second case, we split d in three, so we need to express ψ_d differently, and its transformation matrix is

$$\left[\begin{array}{c|c|c} d_0 & d_1 & d_2 \\ \hline xd_2 & d_0 & d_1 \\ \hline xd_1 & xd_2 & d_0 \end{array}\right] \left(= \left[\begin{array}{c|c|c} d_0 & d_1 & d_2 \\ \hline d_1^\star & d_0 & d_1 \\ \hline d_2^\star & d_1^\star & d_0 \end{array}\right]\right)^{1}. \tag{3.25}$$

   For each diagonal element broken into a self-adjoint matrix $\mathbf{G}_i$ over a smaller ring, we recursively compute its LDL tree as in step 1 and store the result in the left, middle or right child of T (which we denote T.leftchild, T.middlechild and T.rightchild respectively).

[1] The equality in parenthesis is true if and only if d is self-adjoint, i.e. $d^\star = d$.

We continue the recursion until we end up with coefficients in the ring $\mathbb{Q}$ (in the binary case) or $\mathbb{Q}[x]/(x^2 - x + 1)$ (in the ternary case).

A detailed specification of this "LDL tree" strategy is given in the following subsections of this section.

Algorithm 3.26 $\mathsf{LDL}^\star(\mathbf{G})$

Require: A full-rank autoadjoint matrix $\mathbf{G} = (G_{ij}) \in \mathsf{FFT}(\mathbb{Q}[x]/(\phi))^{n \times n}$
Ensure: The LDL* decomposition $\mathbf{G} = \mathbf{LDL}^\star$ over $\mathsf{FFT}(\mathbb{Q}[x]/(\phi))$
Format: All polynomials are in FFT representation.

$\mathbf{L}, \mathbf{D} \leftarrow \mathbf{0}^{n \times n}$
for i from 1 to n **do**
  $L_{ii} \leftarrow 1$
  $D_i \leftarrow G_{ii} - \sum_{j<i} L_{ij} \odot L^\star_{ij} \odot D_j$
  for j from 1 to $i - 1$ **do**
    $L_{ij} \leftarrow \frac{1}{D_j} \left(G_{ij} - \sum_{k<j} L_{ik} \odot L^\star_{jk} \odot D_k \right)$
return $(\mathbf{L}, \mathbf{D})$

The binary case. In the binary case, the application of our LDL tree strategy is rather simple and can be implemented using only the bisection (splitting a polynomial in two). It is specified in algorithm 3.27.

Algorithm 3.27 $\mathsf{ffLDL}^\star(\mathbf{G})$ (Binary case)

Require: A full-rank Gram matrix $\mathbf{G} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n + 1)\right)^{2 \times 2}$
Ensure: A binary tree T
Format: All polynomials are in FFT representation.

$(\mathbf{L}, \mathbf{D}) \leftarrow \mathsf{LDL}^\star(\mathbf{G})$ ▷ $\mathbf{L} = \left[\begin{array}{c|c} 1 & 0 \\ \hline L_{10} & 1 \end{array}\right], \mathbf{D} = \left[\begin{array}{c|c} D_{00} & 0 \\ \hline 0 & D_{11} \end{array}\right]$
T.value $\leftarrow L_{10}$
if $(n = 1)$ **then**
  T.leftchild $\leftarrow D_{00}$
  T.rightchild $\leftarrow D_{11}$
  return T
else
  $d_{00}, d_{01} \leftarrow \mathsf{splitfft}_2(D_{00})$
  $d_{10}, d_{11} \leftarrow \mathsf{splitfft}_2(D_{11})$
  $\mathbf{G}_0 \leftarrow \left[\begin{array}{c|c} d_{00} & d_{01} \\ \hline x d_{01} & d_{00} \end{array}\right], \mathbf{G}_1 \leftarrow \left[\begin{array}{c|c} d_{10} & d_{11} \\ \hline x d_{11} & d_{10} \end{array}\right]$ ▷ $\mathbf{G}_0, \mathbf{G}_1 \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^{n/2} + 1)\right)^{2 \times 2}$
  T.leftchild $\leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_0)$
  T.rightchild $\leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_1)$
  return T

The ternary case. In the binary case, aplying our LDL tree strategy is more complex and also requires the trisection (splitting a polynomial in three). It is specified in algorithm 3.28. In the

Algorithm 3.28 $\mathsf{ffLDL}^\star(\mathbf{G})$ (Ternary case: $\phi = x^n - x^{n/2} + 1$)

Require: A full-rank Gram matrix $\mathbf{G} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n - x^{n/2} + 1)\right)^{k \times k}$, with $k \in \{2, 3\}$
Ensure: A binary tree T
Format: All polynomials are in FFT representation.

$(\mathbf{L}, \mathbf{D}) \leftarrow \mathsf{LDL}^\star(\mathbf{G})$
if $(n > 6)$ **then** ▷ $k = 2$
  $d_{00}, d_{01} \leftarrow \mathsf{splitfft}_2(D_{00})$
  $d_{10}, d_{11} \leftarrow \mathsf{splitfft}_2(D_{11})$
  $\mathbf{G}_0 \leftarrow \left[\begin{array}{c|c} d_{00} & d_{01} \\ \hline x d_{01} & d_{00} \end{array}\right], \mathbf{G}_1 \leftarrow \left[\begin{array}{c|c} d_{10} & d_{11} \\ \hline x d_{11} & d_{10} \end{array}\right]$
  $\mathsf{T}.\mathsf{value} \leftarrow L_{10}$
  $\mathsf{T}.\mathsf{leftchild} \leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_0)$
  $\mathsf{T}.\mathsf{rightchild} \leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_1)$
  return T
if $(n = 6)$ **then** ▷ $k = 2$
  $d_{00}, d_{01}, d_{02} \leftarrow \mathsf{splitfft}_3(D_{00})$
  $d_{10}, d_{11}, d_{12} \leftarrow \mathsf{splitfft}_3(D_{11})$
  $\mathbf{G}_0 \leftarrow \left[\begin{array}{c|c|c} d_{00} & d_{01} & d_{02} \\ \hline x d_{02} & d_{00} & d_{01} \\ \hline x d_{01} & x d_{02} & d_{00} \end{array}\right], \mathbf{G}_1 \leftarrow \left[\begin{array}{c|c|c} d_{10} & d_{11} & d_{12} \\ \hline x d_{12} & d_{10} & d_{11} \\ \hline x d_{11} & x d_{12} & d_{10} \end{array}\right]$
  $\mathsf{T}.\mathsf{value} \leftarrow L_{10}$
  $\mathsf{T}.\mathsf{leftchild} \leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_0)$
  $\mathsf{T}.\mathsf{rightchild} \leftarrow \mathsf{ffLDL}^\star(\mathbf{G}_1)$
  return T
if $(n = 2)$ **then** ▷ $k = 3$
  $\mathsf{T}.\mathsf{value} \leftarrow (L_{10}, L_{20}, L_{21})$
  $\mathsf{T}.\mathsf{leftchild} \leftarrow D_{00}$
  $\mathsf{T}.\mathsf{middlechild} \leftarrow D_{11}$
  $\mathsf{T}.\mathsf{rightchild} \leftarrow D_{22}$
  return T

ternary case, normalization uses a slightly different value for σ. Also, once the normalized leaf value v is computed, it may be relevant to precompute and store $2v/\sqrt{3}$, as the latter value will also be used for signature generation. This is specific to the ternary case.

3.2.3. FFT and NTT

The FFT. Let $f \in \mathbb{Q}[x]/(\phi)$. We note Ω_ϕ the set of complex roots of ϕ. We suppose that ϕ is monic with distrinct roots over $\mathbb{C}$, so that $\phi(x) = \prod_{\zeta \in \Omega_\phi} (x - \zeta)$. We denote by $\mathsf{FFT}_\phi(f)$ the fast Fourier transform of f with respect to ϕ:

$$\mathsf{FFT}_\phi(f) = (f(\zeta))_{\zeta \in \Omega_\phi} \tag{3.26}$$

When ϕ is clear from context, we simply note $\mathsf{FFT}(f)$. We may also use the notation $\hat{f}$ to indicate that $\hat{f}$ is the FFT of f. FFT_ϕ is a ring isomorphism, and we note invFFT_ϕ its inverse. The multiplication in the FFT domain is denoted by $\odot$. We extend the FFT and its inverse to matrices and vectors by component-wise application.

Additions, subtractions, multiplications and divisions of polynomials modulo ϕ can be computed in FFT representations by simply performing them on each coordinate. In particular, this makes multiplications and divisions very efficient.

Of particular interest to us is the FFT for the particular values of ϕ taken in Falcon:

- *Binary case:* $\Omega_\phi = \{\zeta^k | k \in \mathbb{Z}_{2n}^\times\}$, with ζ a primitive $2n$-th complex root of 1.
- *Ternary case:* $\Omega_\phi = \{\zeta^k | k \in \mathbb{Z}_{3n}^\times\}$, with ζ a primitive $3n$-th complex root of 1.

A note on implementing the FFT. There exist several ways of implementing the FFT, which may yield slightly different results. For example, some implementations of the FFT scale our definition by a constant factor (e.g. $1/\deg(\phi)$). Another differentiation point is the order of (the roots of) the FFT. Common orders are the increasing order (i.e. the roots are sorted by their order on the unit circle, starting at 1 and moving clockwise) or (variants of) the bit-reversal order. In the case of Falcon:

- The FFT is not scaled by a constant factor.
- There is no constraint on the order of the FFT, the choice is left to the implementer. However, the chosen order shall be consistent for all the algorithms using the FFT.

Representation of polynomials in algorithms. The algorithms which specify Falcon heavily rely on the fast Fourier transform, and some of them explicitly require that the inputs and/or outputs are given in FFT representation. When the directive "**Format:**" is present at the beginning of an algorithm, it specifies in which format (coefficient or FFT representation) the input/output polynomials shall be represented. When the directive "**Format:**" is absent, no assumption on the format of the input/output polynomials is made.

The NTT. The NTT (Number Theoretic Transform) is the analog of the FFT in the field $\mathbb{Z}_p$, where p is a prime such that $p = 1 \bmod 2n$ (binary case) or $p = 1 \bmod 3n$ (ternary case). Under these conditions, ϕ has exactly n roots (ω_i) over $\mathbb{Z}_p$, and any polynomial $f \in \mathbb{Z}_p[x]/(\phi)$ can be represented by the values $f(\omega_i)$. Conversion to and from NTT representation can be done efficiently

in $O(n \log n)$ operations in $\mathbb{Z}_p$. When in NTT representation, additions, subtractions, multiplications and divisions of polynomials (modulo ϕ and p) can be performed coordinate-wise in $\mathbb{Z}_p$.

In Falcon, the NTT allows for faster implementations of public key operations (using $\mathbb{Z}_q$) and key pair generation (with various medium-sized primes p). Private key operations, though, rely on the fast Fourier sampling, which uses the FFT, not the NTT.

3.2.4. Splitting and Merging

In this section, we make explicit the chains of isomorphisms described in section 3.2.1, by presenting splitting (resp. merging) operators which allow to travel these chains from right to left (resp. left to right).

Let ϕ, ϕ' be cyclotomic polynomials such that we either have $\phi(x) = \phi'(x^2)$ or $\phi(x) = \phi'(x^3)$. In this section we define operators which are at the heart of our signing algorithm. Our algorithms require the ability to split an element of $\mathbb{Q}[x]/(\phi)$ into two or three smaller elements of $\mathbb{Q}[x]/(\phi')$. Conversely, we will require the ability to merge small elements of $\mathbb{Q}[x]/(\phi')$ into a larger element of $\mathbb{Q}[x]/(\phi)$.

3.2.4.1. Bisection: when $\phi(x) = \phi'(x^2)$

In this section we suppose that $\phi(x) = \phi'(x^2)$ and define operators splitting a polynomial $f \in \mathbb{Q}[x]/(\phi)$ in two smaller polynomials of $\mathbb{Q}[x]/(\phi')$, or performing the reciprocal merging operation.

The splitfft$_2$ operator. Let n be the degree of ϕ, and $f = \sum_{i=0}^{n-1} a_i x^i$ be an arbitrary element of $\mathbb{Q}[x]/(\phi)$, f can be decomposed uniquely as $f(x) = f_0(x^2) + x f_1(x^2)$, with $f_0, f_1 \in \mathbb{Q}[x]/(\phi')$. In coefficient representation, such a decomposition is straightforward to write:

$$f_0 = \sum_{0 \le i < n/2} a_{2i} x^i \quad \text{and} \quad f_1 = \sum_{0 \le i < n/2} a_{2i+1} x^i \tag{3.27}$$

f is simply split with respect to its even or odd coefficients. We note $(f_0, f_1) = \mathsf{split}_2(f)$. In Falcon, polynomials are repeatedly split, multiplied together, split again and so forth. To avoid switching back and forth between the coefficient and FFT representation, we always perform the split operation in the FFT representation. It is defined in algorithm 3.29.

Algorithm 3.29 $\mathsf{splitfft}_2(\mathsf{FFT}(f))$

Require: $\mathsf{FFT}(f) = (f(\zeta))_\zeta$ for some $f \in \mathbb{Q}[x]/(\phi)$
Ensure: $\mathsf{FFT}(f_0) = (f_0(\zeta'))_{\zeta'}$ and $\mathsf{FFT}(f_1) = (f_1(\zeta'))_{\zeta'}$ for some $f_0, f_1 \in \mathbb{Q}[x]/(\phi')$
Format: All polynomials are in FFT representation.

1: **for** ζ such that $\phi(\zeta) = 0$ and $\mathrm{Im}(\zeta) > 0$ **do**
2: $\quad \zeta' \leftarrow \zeta^2$
3: $\quad f_0(\zeta') \leftarrow \frac{1}{2}[f(\zeta) + f(-\zeta)]$
4: $\quad f_1(\zeta') \leftarrow \frac{1}{2\zeta}[f(\zeta) - f(-\zeta)]$
5: **return** $(\mathsf{FFT}(f_0), \mathsf{FFT}(f_1))$

$\mathsf{splitfft}_2$ is split_2 realized in the FFT representation: for any f, $\mathsf{FFT}(\mathsf{split}_2(f)) = \mathsf{splitfft}_2(\mathsf{FFT}(f))$. Readers familiar with the Fourier transform will recognize that $\mathsf{splitfft}_2$ is a subroutine of the inverse fast Fourier transform, more precisely the part which from $\mathsf{FFT}(f)$ computes two FFT's twice smaller.

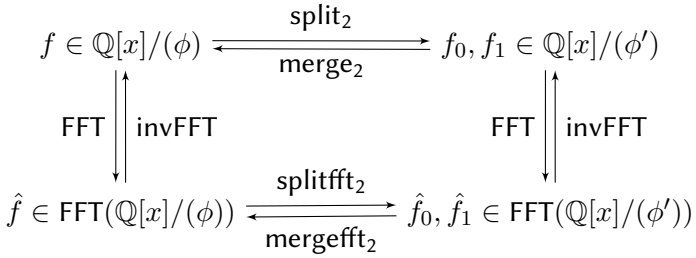


Figure 3.4. – Relationship between FFT, invFFT, split_2, merge_2, $\mathsf{splitfft}_2$ and $\mathsf{mergefft}_2$

The $\mathsf{mergefft}_2$ operator. With the previous notations, we define the operator merge_2 as follows: $\mathsf{merge}_2(f_0, f_1) = f_0(x^2) + xf_1(x^2) \in \mathbb{Q}[x]/(\phi)$. Similarly to split_2, it is often relevant from an efficiently standpoint to perform merge_2 in the FFT representation. This is done in algorithm 3.30.

Algorithm 3.30 $\mathsf{mergefft}_2(f_0, f_1)$

Require: $\mathsf{FFT}(f_0) = (f_0(\zeta'))_{\zeta'}$ and $\mathsf{FFT}(f_1) = (f_1(\zeta'))_{\zeta'}$ for some $f_0, f_1 \in \mathbb{Q}[x]/(\phi')$
Ensure: $\mathsf{FFT}(f) = (f(\zeta))_{\zeta}$ for some $f \in \mathbb{Q}[x]/(\phi)$
Format: All polynomials are in FFT representation.
1: **for** ζ such that $\phi(\zeta) = 0$ **do**
2: $\quad \zeta' \leftarrow \zeta^2$
3: $\quad f(\zeta) \leftarrow f_0(\zeta') + \zeta f_1(\zeta')$
4: **return** $\mathsf{FFT}(f)$

It is immediate that split_2 and merge_2 are inverses of each other, and equivalently $\mathsf{splitfft}_2$ and $\mathsf{mergefft}_2$ are inverses of each other. Just as for $\mathsf{splitfft}_2$, readers familiar with the Fourier transform can observe that $\mathsf{mergefft}_2$ is a step of the fast Fourier transform: it is the reconstruction step which from two small FFT's computes a larger FFT.

Relationship with the FFT. There is no requirement on the order in which the values $f(\zeta)$ (resp. $f_0(\zeta')$, resp. $f_1(\zeta')$) are to be stored, and the choice of this order is left to the implementer. It is however recommended to use a unique order convention for the FFT, invFFT, $\mathsf{splitfft}_2$ and $\mathsf{mergefft}_2$ operators. Since the FFT and invFFT need to implemented anyway, this unique convention can be achieved, e.g. by implementing $\mathsf{splitfft}_2$ as part of invFFT, and $\mathsf{mergefft}_2$ as part of the FFT.

The intricate relationships between the split_2 and merge_2 operators, their counterparts in the FFT representation and the (inverse) fast Fourier transform are illustrated in the commutative diagram of figure 3.4.

3.2.4.2. Trisection: when $\phi(x) = \phi'(x^3)$

In this section we suppose that $\phi(x) = \phi'(x^3)$. In Falcon, this may happen only if ϕ is ternary. We define splitting and merging operators similarly to the bisection case. For any $f \in \mathbb{Q}[x]/(\phi)$, we can uniquely write $f(x) = f_0(x^3) + xf_1(x^3) + x^2 f_2(x^3)$ with $f_0, f_1, f_2 \in \mathbb{Q}[x]/(\phi)$. We then define split_3 as $\mathsf{split}_3(f) = (f_0, f_1, f_2)$, and merge_3 as being the reciprocal operator. Translations of these operators in the FFT domain are given in algorithms 3.31 and 3.32.

Algorithm 3.31 $\mathsf{splitfft}_3(\mathsf{FFT}(f))$

Require: $\mathsf{FFT}(f) = (f(\zeta))_\zeta$ for $f \in \mathbb{Q}[x]/(\phi)$
Ensure: $\mathsf{FFT}(f_0), \mathsf{FFT}(f_1), \mathsf{FFT}(f_2)$ for $f_0, f_1, f_2 \in \mathbb{Q}[x]/(\phi')$
Format: All polynomials are in FFT representation.

1: **for** ζ such that $\phi(\zeta) = 0$ and $\arg(\zeta) \in (0, \frac{2\pi}{3})$ **do**
2: $\quad\zeta' \leftarrow \zeta^3$
3: $\quad f_0(\zeta') \leftarrow \frac{1}{3}\left[f(\zeta) + f(j\zeta) + f(j^2\zeta)\right]$ ▷ j is a primitive cube root of 1: $j = e^{i(2\pi)/3}$
4: $\quad f_1(\zeta') \leftarrow \frac{1}{3\zeta}\left[f(\zeta) + j^2 f(j\zeta) + jf(j^2\zeta)\right]$
5: $\quad f_2(\zeta') \leftarrow \frac{1}{3\zeta^2}\left[f(\zeta) + jf(j\zeta) + j^2 f(j^2\zeta)\right]$
6: **return** $(\mathsf{FFT}(f_0), \mathsf{FFT}(f_1), \mathsf{FFT}(f_2))$

Algorithm 3.32 $\mathsf{mergefft}_3(f_0, f_1, f_2)$

Require: $\mathsf{FFT}(f_0), \mathsf{FFT}(f_1), \mathsf{FFT}(f_2)$ for $f_0, f_1, f_2 \in \mathbb{Q}[x]/(\phi')$
Ensure: $\mathsf{FFT}(f) = (f(\zeta))_\zeta$ for $f \in \mathbb{Q}[x]/(\phi)$
Format: All polynomials are in FFT representation.

1: **for** ζ such that $\phi(\zeta) = 0$ **do**
2: $\quad\zeta' \leftarrow \zeta^3$
3: $\quad f(\zeta) \leftarrow f_0(\zeta') + \zeta f_1(\zeta') + \zeta^2 f_2(\zeta')$
4: $\quad f(j\zeta) \leftarrow f_0(\zeta') + j\zeta f_1(\zeta') + j^2\zeta^2 f_2(\zeta')$
5: $\quad f(j^2\zeta) \leftarrow f_0(\zeta') + j^2\zeta f_1(\zeta') + j\zeta^2 f_2(\zeta')$
6: **return** $\mathsf{FFT}(f)$

Relationship with the FFT. The operators split_3, merge_3, $\mathsf{splitfft}_3$ and $\mathsf{mergefft}_3$ have an identical relationship with the FFT as their bisection counterparts do (see section 3.2.4.1 and figure 3.4).

3.2.4.3. The special case $\phi(x) = x^2 - x + 1$

The roots of the polynomial $x^2 - x + 1$ are the two sixth roots of unity $\zeta_6 = \frac{1}{2} + \frac{\sqrt{3}}{2}i$ and $\bar{\zeta_6} = \frac{1}{2} - \frac{\sqrt{3}}{2}i$. In contrast to the other cases it is not true that for a root ζ also $-\zeta$ is a root of $x^2 - x + 1$. We give specialized split and merge algorithms for this case in Algorithms 3.33 and 3.34. Let $f = f_0 + f_1$ be a polynomial in $\mathbb{Q}[x]/(x^2 - x + 1)$. Then its FFT representation is given by $f(\zeta_6) = f_0 + \frac{1}{2}f_1 + \frac{\sqrt{3}}{2}f_1 i$ and the complex conjugate. The splitted polynomials f_0 and f_1 lie in $\mathbb{Q}$ and hence their FFT representation is just f_0 and f_1. We have $f_1 = \frac{2}{\sqrt{3}}\Im f(\zeta_6)$ and $f_0 = \Re f(\zeta_6) - \frac{1}{2}f_1$.

Algorithm 3.33 $\mathsf{splitfft}_6(\mathsf{FFT}(f))$

Require: $\mathsf{FFT}(f) = (f(\zeta))_\zeta$ for $f \in \mathbb{Q}[x]/(x^2 - x + 1)$
Ensure: $f_0, f_1 \in \mathbb{Q}$
Format: All polynomials are in FFT representation.

1: $\zeta \leftarrow \frac{1}{2} + \frac{\sqrt{3}}{2}i$ ▷ sixth root of unity
2: $f_1 \leftarrow \frac{2}{\sqrt{3}}\Im(f(\zeta))$ ▷ $\Im$ denotes the imaginary part
3: $f_0 \leftarrow \Re(f(\zeta)) - \frac{1}{2}f_1$ ▷ $\Re$ denotes the real part
4: **return** (f_0, f_1)

Algorithm 3.34 $\mathsf{mergefft}_6(f_0, f_1)$

Require: $f_0, f_1 \in \mathbb{Q}$
Ensure: $\mathsf{FFT}(f) = (f(\zeta))_\zeta$ for $f \in \mathbb{Q}[x]/(x^2 - x + 1)$
Format: All polynomials are in FFT representation.
1: **for** ζ such that $\zeta^2 - \zeta + 1 = 0$ **do**
2: $\quad f(\zeta) \leftarrow f_0 + \zeta f_1$
3: **return** $\mathsf{FFT}(f)$

3.2.4.4. Algebraic interpretation

The purpose of the splitting and merging operators that we defined is not only to represent elements of $\mathbb{Q}[x]/(\phi)$ using smaller elements of $\mathbb{Q}[x]/(\phi')$, but to do so in a manner which is compatible with ring operations. As an illustration, we consider the operation:

$$a = bc \tag{3.28}$$

where $a, b, c \in \mathbb{Q}[x]/(\phi)$. For $f \in \mathbb{Q}[x]/(\phi)$, we consider the associated endomorphism $\psi_f : z \in \mathbb{Q}[x]/(\phi) \mapsto fz$. The equation 3.28 can be rewritten as $a = \psi_c(b)$. We will show how to use the splitting operator to express it as a vector-matrix product in the module $(\mathbb{Q}[x]/(\phi'))^k$.

1. *Bisection.* By the splitting isomorphism split_2, a and b (resp. ψ_c) can also be considered as elements (resp. an endomorphism) of $(\mathbb{Q}[x]/(\phi'))^2$. The equation 3.28 can be expressed over $\mathbb{Q}[x]/(\phi')$ as

$$\left[\begin{array}{c|c} a_0 & a_1 \end{array}\right] = \left[\begin{array}{c|c} b_0 & b_1 \end{array}\right] \left[\begin{array}{c|c} c_0 & c_1 \\ \hline xc_1 & c_0 \end{array}\right] \tag{3.29}$$

2. *Trisection.* The equation 3.28 is now expressed as

$$\left[\begin{array}{c|c|c} a_0 & a_1 & a_2 \end{array}\right] = \left[\begin{array}{c|c|c} b_0 & b_1 & b_2 \end{array}\right] \left[\begin{array}{c|c|c} c_0 & c_1 & c_2 \\ \hline xc_2 & c_0 & c_1 \\ \hline xc_1 & xc_2 & c_0 \end{array}\right] \tag{3.30}$$

More formally, we have used the fact that splitting operators are isomorphisms between $\mathbb{Q}[x]/(\phi)$ and $(\mathbb{Q}[x]/(\phi'))^k$, which express elements of $\mathbb{Q}[x]/(\phi)$ in the $(\mathbb{Q}[x]/(\phi'))$-basis $\{1, x\}$ for the bisection, or $\{1, x, x^2\}$ for the trisection (hence "breaking" a, b in vectors of smaller elements).

Similarly, writing the transformation matrix of the endomorphism ψ_c in the basis $\{1, x\}$ (resp. $\{1, x, x^2\}$) yields the 2×2 (resp. 3×3) matrix of the equation 3.29 (resp. 3.30)

3.2.4.5. Relationship with the field norm

The splitting and merging operators allow to easily express the field norm for some specific cyclotomic fields. Let $\mathbb{L} = \mathbb{Q}[x]/(\phi), \mathbb{K} = \mathbb{Q}[x]/(\phi')$ and $f \in \mathbb{L}$. Since by definition $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \det_{\mathbb{K}}(\psi_d)$, we can use the equations 3.29 and 3.30 to compute it explicitly. This yields:

1. If $\phi'(x^2) = \phi(x)$, then $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = f_0^2 - xf_1^2$, where $(f_0, f_1) = \mathsf{split}_2(f)$;
2. If $\phi'(x^3) = \phi(x)$, then $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = f_0^3 + xf_1^3 + x^2f_2^3 + 3f_0f_1f_2$, where $(f_0, f_1, f_2) = \mathsf{split}_3(f)$;

3. If $\phi(x) = x^2 - x + 1$, then $\mathrm{N}_{\mathbb{L}/\mathbb{Q}}(f) = f_0^2 + f_0 f_1 + f_1^2$, where $f = f_0 + x f_1$.

For $f \in \mathbb{L}$ with $\mathbb{L} = \mathbb{Q}[x]/(\phi)$, we will also denote $\mathrm{N}(f) = \mathrm{N}_{\mathbb{L}/\mathbb{K}}(f)$, where:

1. if $\exists$ a cyclotomic polynomial ϕ' such that $\phi'(x^2) = \phi(x)$, then $\mathbb{K} = \mathbb{Q}[x]/(\phi')$;
2. else, if $\exists$ a cyclotomic polynomial ϕ such that $\phi'(x^3) = \phi(x)$, then $\mathbb{K} = \mathbb{Q}[x]/(\phi')$;
3. else, $\mathbb{K} = \mathbb{Q}$.

For the values of ϕ considered in this document, this allows to define $\mathrm{N}(f)$ in an unambiguous way for any $f \in \mathbb{Q}[x]/(\phi)$.

3.2.5. Hashing

As for any hash-and-sign signature scheme, the first step to sign a message or verify a signature consists of hashing the message. In our case, the message needs to be hashed into a polynomial in $\mathbb{Z}_q[x]/(\phi)$. An approved extendable-output hash function (XOF), as specified in FIPS 202 [NIS15], shall be used during this procedure.

This XOF shall have a security level at least equal to the security level targeted by our signature scheme. In addition, we should be able to start hashing a message without knowing the security level at which it will be signed. For these reasons, we use a unique XOF for all security levels: SHAKE-256.

- SHAKE-256 -Init () denotes the initialization of a SHAKE-256 hashing context;
- SHAKE-256 -Inject (ctx, str) denotes the injection of the data str in the hashing context ctx;
- SHAKE-256 -Extract (ctx, b) denotes extraction from a hashing context ctx of b bits of pseudo-randomness.

In Falcon, big-endian convention is used to interpret a chunk of b bits, extracted from a SHAKE-256 instance, into an integer in the 0 to $2^b - 1$ range (the first of the b bits has numerical weight 2^{b-1}, the last has weight 1).

Algorithm 3.35 HashToPoint(str, q, n)

Require: A string str, a modulus $q \leq 2^{16}$, a degree $n \in \mathbb{N}^\star$
Ensure: An polynomial $c = \sum_{i=0}^{n-1} c_i x^i$ in $\mathbb{Z}_q[x]$

```
k ← ⌊2^16/q⌋
ctx ← SHAKE-256-Init()
SHAKE-256-Inject(ctx, str)
i ← 0
while i < n do
    t ← SHAKE-256-Extract(ctx, 16)
    if t < kq then
        c_i ← t mod q
        i ← i + 1
return c
```

Algorithm 3.35 defines the hashing process used in Falcon. It is defined for any $q \leq 2^{16}$.

Possible variants.

- If $q > 2^{16}$, then larger chunks can be extracted from SHAKE-256 at each step.
- Algorithm 3.35 may be difficult to efficiently implement in a constant-time way; constant-timeness may be a desirable feature if the signed data is also secret.

  A variant which is easier to implement with constant-time code extracts 64 bits instead of 16 at step 6, and omits the conditional check of step 7. While the omission of the check means that some target values modulo q will be slightly more probable than others, a Rényi argument [Pre17] allows to claim that this variant is secure for the parameters set by NIST [NIS16].

Of course, any variant deviating from the procedure expressed in algorithm 3.35 implies that the same message will hash to a different value, which breaks interoperability.

3.2.6. Signature Generation

At a high level, the principle of the signature generation algorithm is simple: it first computes a hash value $c \in \mathbb{Z}_q[x]/(\phi)$ from the message m and a salt r, and it then uses its knowledge of the secret key f, g, F, G to compute two short values s_1, s_2 such that $s_1 + s_2 h = c \bmod q$.

A naive way to find such short values (s_1, s_2) would be to compute $\mathbf{t} \leftarrow (c, 0) \cdot \mathbf{B}^{-1}$, to round it coefficient-wise to a vector $\mathbf{z}$ and to output $(s_1, s_2) \leftarrow (\mathbf{t} - \mathbf{z})\mathbf{B}$; one can check that (s_1, s_2) does indeed fill all the requirements to be a legitimate, but this method is known to be insecure and to leak the secret key.

The proper way to generate (s_1, s_2) without leaking the secret key is to use a trapdoor sampler (see section 3.1.2 for a brief reminder on trapdoor samplers). In Falcon, we use a trapdoor sampler called fast Fourier sampling. The computation of the falcon tree T by the procedure ffLDL* during the key pair generation was the initialization step of this trapdoor sampler.

The heart of our signature generation, the procedure ffSampling (algorithm 3.37), will adaptatively apply a randomized rounding (according to a discrete Gaussian distribution) on the coefficients of $\mathbf{t}$. But it will do so in an adaptative manner, using the information stored in the Falcon tree T.

At a high level, our fast Fourier sampling algorithm can be seen as a recursive variant of Klein's well known trapdoor sampler (also known as the GPV sampler). Klein's sampler uses a matrix $\mathbf{L}$ (and the norm of Gram-Schmidt vectors) as a trapdoor, whereas ours uses a tree of such matrices (or rather, a tree of their non-trivial elements). Given $\mathbf{t} = (t_0, t_1)$, our algorithm first splits t_1 using the splitting operator, recursively applies itself to it (using the right child T.rightchild of T), and uses the merging operator to lift the solution to the base ring of $\mathbb{Z}[x]/(\phi)$; it then applies itself again recursively with t_0. It is important to notice that the recursions cannot be done in parallel: the second recursion takes into account the result of the first recursion, and this is done using information contained in T.value.

The most delicate part of our signature algorithm is the fast Fourier sampling described in algorithm 3.37, because it makes use of the Falcon tree and of discrete Gaussians over $\mathbb{Z}$. The rest of the algorithm, including the compression of the signature, is rather straightforward to implement.

Formally, given a secret key sk and a message m, the signer uses sk to sign m as follows:

1. A random salt r is generated uniformly in $\{0, 1\}^{320}$. The concatenated string $(\mathsf{r} \| \mathsf{m})$ is then hashed to a point $c \in \mathbb{Z}_q[x]/(\phi)$ as specified by algorithm 3.35
2. A (not necessarily short) preimage $\mathbf{t}$ of c is computed, and is then given as input to the fast Fourier sampling algorithm, which outputs two short polynomials $s_1, s_2 \in \mathbb{Z}[x]/(\phi)$ (in FFT representation) such that $s_1 + s_2 h = c \bmod q$, as specified by algorithm 3.37.

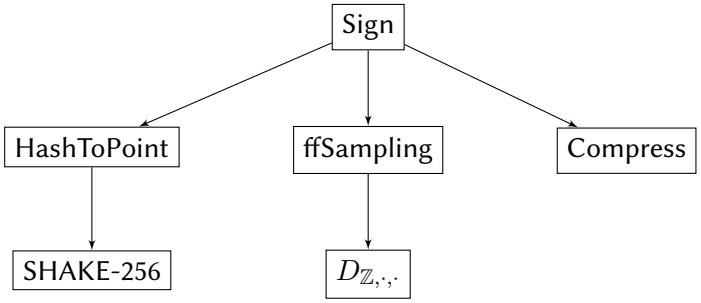


Figure 3.5. – Flowchart of the signature

3. s_2 is encoded (compressed) to a bitstring s as specified in section 3.2.8.

4. The signature consists of the pair (r, s).

A note on sampling over $\mathbb{Z}$. The algorithm 3.37 requires access to an oracle $\mathcal{D}$ for the distribution $D_{\mathbb{Z},\sigma',c'}$, where σ' can be the value of any leaf of the private FALCON tree T, and $c' \in \mathbb{Q}$ is arbitrary[2]. How to implement $\mathcal{D}$ is outside the scope of this chapter. It is only required that the Rényi divergence between this oracle and an ideal discrete Gaussian $D_{\mathbb{Z},\sigma',c'}$ verifies $R_{512}(\mathcal{D}\|D_{\mathbb{Z},\sigma',c'}) \leq 1 + 2^{-66}$, for the definition of the Rényi divergence given in e.g. [BLL+15]. It is noteworthy that the range of possible values for the standard deviation in the Gaussian sampler is limited: it is always greater than 1.2, and always lower than 1.9 (in the binary case) or 2.20 (in the ternary case). The FALCON reference implementation uses a Gaussian sampler based on rejection sampling against a bimodal distribution.

The general architecture of the signing procedure is illustrated in figure 3.5.

Algorithm 3.36 Sign (m, sk, β)

Require: A message m, a secret key sk, a bound β
Ensure: A signature sig of m
1: $\mathsf{r} \leftarrow \{0,1\}^{320}$ uniformly
2: $c \leftarrow \mathsf{HashToPoint}(\mathsf{r}\|\mathsf{m})$
3: $\mathbf{t} \leftarrow (\mathsf{FFT}(c), \mathsf{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$
4: **do**
5: $\quad \mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathsf{T})$
6: $\quad \mathbf{s} = (\mathbf{t} - \mathbf{z})\hat{\mathbf{B}}$
7: **while** $\|\mathbf{s}\| > \beta$
8: $(s_1, s_2) \leftarrow \mathsf{invFFT}(\mathbf{s})$
9: $\mathsf{s} \leftarrow \mathsf{Compress}(s_2)$
10: **return** $\mathsf{sig} = (\mathsf{r}, \mathsf{s})$

[2] In the ternary case, leaf values may also be multiplied by $2/\sqrt{3}$.

3.2.6.1. Fast Fourier sampling: the binary case

This section describes our fast Fourier sampling algorithm in the binary case. This is done in algorithm 3.37. It is worth noticing that we perform all the operations in FFT representation for efficiency reasons, but the whole algorithm could also be executed in coefficient representation instead, at a price of a $O(n/\log n)$ penalty in speed.

Algorithm 3.37 $\mathsf{ffSampling}_n(\mathbf{t}, \mathsf{T})$ (Binary case)

Require: $\mathbf{t} = (t_0, t_1) \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n+1)\right)^2$, a Falcon tree T
Ensure: $\mathbf{z} = (z_0, z_1) \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^n+1)\right)^2$
Format: All polynomials are in FFT representation.

if $n = 1$ **then**
  $\sigma' \leftarrow \mathsf{T.value}$
  $z_0 \leftarrow D_{\mathbb{Z},\, t_0,\, \sigma'}$ ▷ Since $n = 1$, $t_0 = \mathsf{invFFT}(t_0) \in \mathbb{Q}$ and $z_0 = \mathsf{invFFT}(z_0) \in \mathbb{Z}$
  $z_1 \leftarrow D_{\mathbb{Z},\, t_1,\, \sigma'}$ ▷ Since $n = 1$, $t_1 = \mathsf{invFFT}(t_1) \in \mathbb{Q}$ and $z_1 = \mathsf{invFFT}(z_1) \in \mathbb{Z}$
  return $\mathbf{z} = (z_0, z_1)$
$(\ell, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
$\mathbf{t}_1 \leftarrow \mathsf{splitfft}_2(t_1)$ ▷ $\mathbf{t}_1 \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^{n/2}+1)\right)^2$
$\mathbf{z}_1 \leftarrow \mathsf{ffSampling}_{n/2}(\mathbf{t}_1, \mathsf{T}_1)$ ▷ First recursive call to $\mathsf{ffSampling}_{n/2}$
$z_1 \leftarrow \mathsf{mergefft}_2(\mathbf{z}_1)$ ▷ $\mathbf{z}_1 \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^{n/2}+1)\right)^2$
$t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \ell$
$\mathbf{t}_0 \leftarrow \mathsf{splitfft}_2(t'_0)$
$\mathbf{z}_0 \leftarrow \mathsf{ffSampling}_{n/2}(\mathbf{t}_0, \mathsf{T}_0)$ ▷ Second recursive call to $\mathsf{ffSampling}_{n/2}$
$z_0 \leftarrow \mathsf{mergefft}_2(\mathbf{z}_0)$
return $\mathbf{z} = (z_0, z_1)$

3.2.6.2. Fast Fourier sampling: the ternary case

This section describes our fast Fourier sampling algorithm in the ternary case. This is done in algorithm 3.38. Once again, the description is more complex than in the binary case but the general strategy remains the same.

We explain the last step for $n = 2$. The scalar product on $\mathbb{Q}^2$ induced by the isomorphism $\mathbb{Q}[X]/(x^2 - x + 1) \cong \mathbb{Q}^2$ is

$$\langle (a_0, a_1), (b_0, b_1) \rangle = a_0 b_0 + a_1 b_1 + \frac{1}{2} a_0 b_1 + \frac{1}{2} a_1 b_0.$$

Let $t = t_0 + t_1 x \in \mathbb{Q}[x]/(x^2 - x + 1)$. Then the matrix over $\mathbb{Q}$ corresponding to the endomorphism $\psi_t : z \mapsto tz$ is given by

$$\left[\begin{array}{c|c} t_0 & t_1 \\ \hline -t_1 & t_0 + t_1 \end{array}\right].$$

We need to Gram-Schmidt orthogonalize this matrix with respect to the norm above. One finds for the Gram-Schmidt coefficient:

$$\ell = \frac{\langle (-t_1, t_0 + t_1), (t_0, t_1) \rangle}{\langle (t_0, t_1), (t_0, t_1) \rangle} = \frac{1}{2}.$$

The lengths of the Gram-Schmidt vectors are $\|(t_0, t_1)\|$ and $\|(-t_1 - \frac{1}{2}t_0, t_0 + \frac{1}{2}t_1)\| = \frac{\sqrt{3}}{2}\|(t_0, t_1)\|$. In the LDL^* decomposition of the basis matrix over $\mathbb{Q}[x]/(x^2 - x + 1)$, the diagonal coefficients of D are precisely the norms $\|t\|^2$.

Algorithm 3.38 ffSampling $_n(\mathbf{t}, \mathsf{T})$ (Ternary case)

Require: $\mathbf{t} = (t_0, t_1) \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n - x^{n/2} + 1)\right)^k$, a Falcon tree T
Ensure: $\mathbf{z} = (z_0, z_1) \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^n - x^{n/2} + 1)\right)^k$
Format: All polynomials are in FFT representation.

if $(n > 6)$ **then** ▷ $k = 2$
  $(\ell, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
  $z_1 \leftarrow \mathsf{mergefft}_2 \circ \mathsf{ffSampling}_{n/2}(\mathsf{splitfft}_2(t_1), \mathsf{T}_1)$
  $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \ell$
  $z_0 \leftarrow \mathsf{mergefft}_2 \circ \mathsf{ffSampling}_{n/2}(\mathsf{splitfft}_2(t'_0), \mathsf{T}_0)$
  return $\mathbf{z} = (z_0, z_1)$
if $(n = 6)$ **then** ▷ $k = 2$
  $(\ell, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
  $z_1 \leftarrow \mathsf{mergefft}_3 \circ \mathsf{ffSampling}_{n/3}(\mathsf{splitfft}_3(t_1), \mathsf{T}_1)$
  $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \ell$
  $z_0 \leftarrow \mathsf{mergefft}_3 \circ \mathsf{ffSampling}_{n/3}(\mathsf{splitfft}_3(t'_0), \mathsf{T}_0)$
  return $\mathbf{z} = (z_0, z_1)$
if $(n = 2)$ **then** ▷ $k = 3$
  $((\ell_{10}, \ell_{20}, \ell_{21}), \mathsf{T}_0, \mathsf{T}_1, \mathsf{T}_2) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.middlechild}, \mathsf{T.rightchild})$
  $z_2 \leftarrow \mathsf{mergefft}_6 \circ \mathsf{ffSampling}_{n/2}(\mathsf{splitfft}_6(t_2), \mathsf{T}_2)$
  $t'_1 \leftarrow t_1 + (t_2 - z_2) \odot \ell_{21}$
  $z_1 \leftarrow \mathsf{mergefft}_6 \circ \mathsf{ffSampling}_{n/2}(\mathsf{splitfft}_6(t'_1), \mathsf{T}_1)$
  $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \ell_{10} + (t_2 - z_2) \odot \ell_{20}$
  $z_0 \leftarrow \mathsf{mergefft}_6 \circ \mathsf{ffSampling}_{n/2}(\mathsf{splitfft}_6(t'_1), \mathsf{T}_0)$
  return $\mathbf{z} = (z_0, z_1, z_2)$
if $(n = 1)$ **then** ▷ $k = 2$
  $\sigma' \leftarrow \mathsf{T.value}$
  $z_1 \leftarrow D_{\mathbb{Z},\, t_1,\, \frac{2}{\sqrt{3}}\sigma'}$
  $t'_0 \leftarrow t_0 + \frac{1}{2}(t_1 - z_1)$
  $z_0 \leftarrow D_{\mathbb{Z},\, t'_0,\, \sigma'}$
  return $\mathbf{z} = (z_0, z_1)$

Note that in the ternary case, for each tree leaf value σ', the Gaussian sampler is invoked twice, with standard deviations σ' and $2\sigma'/\sqrt{3}$. The practical consequence is that the range of inputs for the Gaussian sampler is larger in the ternary case, compared to the binary case.

3.2.7. Signature Verification

The signature verification procedure is much simpler than the key pair generation and the signature generation, both to describe and to implement. Given a public key $\mathsf{pk} = h$, a message m, a signature sig = (r,s) and an acceptance bound β, the verifier uses pk to verify that sig is a valid signature for the message m as specified hereinafter:

1. The value r (called "the salt") and the message m are concatenated to a string $(\mathsf{r}\|\mathsf{m})$ which is hashed to a polynomial $c \in \mathbb{Z}_q[x]/(\phi)$ as specified by algorithm 3.35.
2. s is decoded (decompressed) to a polynomial $s_2 \in \mathbb{Z}[x]/(\phi)$ as specified in section 3.2.8.
3. The value $s_1 = c - s_2 h \bmod q$ is computed.
4. If $\|(s_1, s_2)\| \leq \beta$, then the signature is accepted as valid. Otherwise, it is rejected.

The only subtlety here is that, as recalled in the notations, $\|\|$ denotes the embedding norm and not the coefficient norm. However, it is possible to compute it in linear time. Given two polynomials a and b in $\mathbb{Z}_q[x]/(\phi)$, whose coefficients are denoted a_j and b_j, respectively, the norm $\|(a, b)\|$ is such that, in the binary case:

$$\|(a, b)\|^2 = \sum_{j=0}^{n-1} (a_j^2 + b_j^2) \tag{3.31}$$

and, in the ternary case:

$$\|(a, b)\|^2 = \sum_{j=0}^{n-1} (a_j^2 + b_j^2) + \sum_{j=0}^{n/2-1} (a_j a_{j+n/2} + b_j b_{j+n/2}) \tag{3.32}$$

The signature verification is described in algorithm 3.39.

Algorithm 3.39 Vf(m, sig, pk, β)

Require: A message m, a signature sig $= (\mathsf{r}, \mathsf{s})$, a public key $\mathsf{pk} = h \in \mathbb{Z}_q[x]/(\phi)$, a bound β
Ensure: Accept or reject

```
c ← HashToPoint(r‖m, q, n)
s2 ← Decompress(s)
s1 ← c − s2·h mod q
if ‖(s1, s2)‖ ≤ β then
    accept
else
    reject
```

Computation of s_1 can be performed entirely in $\mathbb{Z}_q[x]/(\phi)$; the resulting values should then be normalized to the $\lceil -q/2 \rceil$ to $\lfloor q/2 \rfloor$ range.

In order to avoid computing a square root, the squared norm can be computed, using only integer operations, and then compared to β^2.

3.2.8. Encoding Formats

3.2.8.1. Bits and Bytes

A *byte* is a sequence of eight bits (formally, an *octet*). Within a byte, bits are ordered from left to right. A byte has a numerical value, which is obtained by adding the weighted bits; the leftmost bit, also called “top bit” or “most significant”, has weight 128; the next bit has weight 64, and so on, until the rightmost bit, which has weight 1.

Some of the encoding formats defined below use sequences of bits. When a sequence of bits is represented as bytes, the following rules apply:

- The first byte will contain the first eight bits of the sequence; the second byte will contain the next eight bits, and so on.

- Within each byte, bits are ordered left-to-right in the same order as they appear in the source bit sequence.

- If the bit sequence length is not a multiple of 8, up to 7 extra padding bits are added at the end of the sequence. The extra padding bits MUST have value zero.

This handling of bits matches widely deployed standard, e.g. bit ordering in the SHA-2 and SHA-3 functions, and `BIT STRING` values in ASN.1.

3.2.8.2. Compressing Gaussians

In Falcon as in other lattice-based signatures schemes, it is not uncommon to have to deal with discrete Gaussians. In particular, the signature of a message essentially consists of a polynomial $s \in \mathbb{Z}_q[x]/(\phi)$ which coefficients are distributed around 0 according to a discrete Gaussian distribution of standard deviation $\sigma = 1.55\sqrt{q} \ll q$. A naive encoding of s would require about $\lceil \log_2 q \rceil \cdot \deg(\phi)$ bits, which is far from optimal for communication complexity.

In this section we specify algorithms for compressing and decompressing efficiently polynomials such as s. The description of this compression procedure is simple:

1. For each coefficient s_i, a compressed string str_i is defined as follows:

   a) The first bit of str_i is the sign of s_i;

   b) The 7 next bits of str_i are the 7 least significant bits of $|s_i|$, in order of significance, i.e. most to least significant (in the ternary case, we use 8 bits here, owing to the larger value of q);

   c) The last bits of str_i are an encoding of the most significant bits of $|s_i|$ using unary coding. If $\lfloor |s_i|/2^7 \rfloor = k$, then its encoding is $\underbrace{0 \ldots 0}_{k \text{ times}} 1$;

2. The compression of s is the concatenated string $\text{str} \leftarrow (\text{str}_0 \| \text{str}_1 \| \ldots \| \text{str}_{n-1})$.

The rationale behind this encoding is based on two observations. First, since $s_i \bmod 2^7$ is close to uniform, there is nothing to be gained by trying to compress the 7 least significant bits of s_i. Second, if a Huffman table is computed for the most significant bits of $|s_i|$, it results in the unary code we just described. So our unary code is actually a Huffman code for the distribution of the most significant bits of $|s_i|$. A formal description is given in algorithm 3.40, for the binary case. For the ternary case, the same algorithm is used, except that 8 low bits are used instead of 7.

Algorithm 3.40 Compress(s) (Binary case)

Require: A polynomial $s = \sum s_i x^i \in \mathbb{Z}[x]$ of degree $< n$
Ensure: A compressed representation str of s

str $\leftarrow \{\}$ ▷ str is the empty string
for i from 0 to $n-1$ **do** ▷ At each step, str $\leftarrow$ (str$\|$str$_i$)
  str $\leftarrow$ (str$\|b$), where $b = 1$ if $s_i > 0$, $b = 0$ otherwise
  str $\leftarrow$ (str$\|b_0 b_1 \dots b_6$), where $b_j = \lfloor |s_i|/2^j \rfloor \bmod 2$
  $k \leftarrow \lfloor |s_i|/2^7 \rfloor$
  for j from 1 to k **do**
    str $\leftarrow$ (str$\|0$)
  str $\leftarrow$ (str$\|1$)
return str

The corresponding decompression algorithm is given in algorithm 3.41. There again, the ternary case is similar, except that it expects 8 low bits instead of 7. For any polynomial $s \in \mathbb{Z}[x]$, it holds that Decompress $\circ$ Compress$(s) = s$.

Algorithm 3.41 Decompress(str) (Binary case)

Ensure: A string str $= (\text{str}[i])_{i=0\dots\ell-1}$ of length ℓ
Require: A polynomial $s = \sum s_i x^i \in \mathbb{Z}[x]$

$j \leftarrow 0$
for i from 0 to $n-1$ **do**
  $s'_i \leftarrow \sum_{j=0}^{6} 2^j \text{str}[1+j]$ ▷ We recover the lowest bits of $|s_i|$.
  $k \leftarrow 0$ ▷ We recover the highest bits of $|s_i|$.
  while str$[7+k] = 0$ **do**
    $k \leftarrow k + 1$
  $s_i \leftarrow (-1)^{\text{str}[0]+1} \cdot (s'_i + 2^7 k)$ ▷ We recover s_i.
  str $\leftarrow$ str$[9+k \dots \ell-1]$ ▷ We remove the bits of str already read.
return $s = \sum_{i=0}^{n-1} s_i x^i$

3.2.8.3. Signatures

A Falcon signature consists of two strings r and s. They are normally encoded separately, because the salt r must be known *before* beginning to hash the message itself, while the s value can be obtained or verified only after the whole message has been processed. In a format that supports streamed processing of long messages, the salt r would normally be encoded before the message, while the s value would appear after the message bytes.

s encodes the polynomial s_2 in a sequence of bytes. The first byte has the following format (bits indicated from most to least significant):

```
t c c 0 n n n n
```

with these conventions:

- The leftmost bit `t` is 1 in the ternary case, 0 in the binary case.

- Bits cc indicates the compression algorithm: 00 is "uncompressed", and 01 is "compressed". Values 10 and 11 are reserved and shall not be used for now.
- The fourth bit is reserved and must be zero.
- Bits nnnn encode a value ℓ. In the binary case, degree is $n = 2^\ell$; in the ternary case, degree is $n = 3 \cdot 2^{\ell-1}$. Degree must be in the allowed range (2 to 1024 in binary, 12 to 768 in ternary).

Following this header byte is the encoded s_2 value (s string). If the compression algorithm is "uncompressed", then the n coefficients of s_2 follow, in signed (two's complement) big-endian 16-bit encoding. If the algorithm is "compressed", then the compression algorithm described in section 3.2.8.2 is applied and yields s as a sequence of bits; extra bits in the final byte (if the length of s is not a multiple of 8) are set to 0.

3.2.8.4. Public Keys

A Falcon public key is a polynomial h whose coefficients are considered modulo q. An encoded public key starts with a header byte:

```
t 0 0 0 n n n n
```

with these conventions:

- The leftmost bit t is 1 in the ternary case, 0 in the binary case.
- The next three bits are reserved and must be zero.
- Bits nnnn encode a value ℓ. In the binary case, degree is $n = 2^\ell$; in the ternary case, degree is $n = 3 \cdot 2^{\ell-1}$. Degree must be in the allowed range (2 to 1024 in binary, 12 to 768 in ternary).

After the header byte comes the encoding of h: each value (in the 0 to $q-1$ range) is encoded as a 14-bit or 15-bit sequence (in the binary case, $q = 12289$ and 14 bits per value are used; in the ternary case, $q = 18433$ and 15 bits are used). The encoded values are concatenated into a bit sequence of $14n$ or $15n$ bits, which is then represented as $\lceil 14n/8 \rceil$ or $\lceil 15n/8 \rceil$ bytes.

3.2.8.5. Private Keys

Private keys use the following header byte:

```
t c c g n n n n
```

with these conventions:

- The leftmost bit t is 1 in the ternary case, 0 in the binary case.
- Bits cc indicates the compression algorithm: 00 is "uncompressed", and 01 is "compressed". Values 10 and 11 are reserved and shall not be used for now.
- Bit g is 0 if the key includes the polynomial G, or 1 if G is absent.
- Bits nnnn encode a value ℓ. In the binary case, degree is $n = 2^\ell$; in the ternary case, degree is $n = 3 \cdot 2^{\ell-1}$. Degree must be in the allowed range (2 to 1024 in binary, 12 to 768 in ternary).

Following the header byte are the encodings of f, g, F, and optionally G, in that order. When no compression is used (bit c is 0), each coordinate is encoded as a 16-bit signed value (two's complement, big-endian convention). When compression is used, each polynomial is compressed with the algorithm described in section 3.2.8.2; each of the four polynomial yields a bit sequence which is split into bytes with up to 7 padding bits so that each encoded polynomial starts at a byte boundary.

When G is absent (bit g is 1), users must recompute it. This is easily done thanks to the NTRU equation:

$$G = (q + gF)/f \mod \phi \tag{3.33}$$

Since the coefficients of f, g, F and G are small, this computation can be done modulo q as well, using the same techniques as signature verification (e.g. the NTT).

3.2.9. Recommended Parameters

In this section, we specify three set of parameters to address the five security levels required by NIST [NIS16, Section 4.A.5]. These can be found in table 3.2.

Level	Dimension n	Polynomial ϕ	Modulus q	Acceptance bound β^2
1 - AES128	512	$x^n + 1$	12289	43533782
2 - SHA256 3 - AES192	768	$x^n - x^{n/2} + 1$	18433	100464491
4 - SHA384 5 - AES256	1024	$x^n + 1$	12289	87067565

Table 3.2. – Falcon security parameters

Acceptance bound. It is important that signers and verifiers agree *exactly* on the acceptance bound, since signatures may come arbitrarily close to that bound (signers restart the signing process when they exceed it). We thus define the bound β in the binary case (with $q = 12289$) such that:

$$\beta^2 = \left\lfloor \frac{87067565n}{1024} \right\rfloor \tag{3.34}$$

and, in the ternary case (with $q = 18433$):

$$\beta^2 = \left\lfloor \frac{100464491n}{768} \right\rfloor \tag{3.35}$$

3.3. Implementation and Performances

We list here a number of noteworthy points related to implementation.

3.3.1. Floating-Point

Signature generation, and also part of key pair generation, involve the use of complex numbers. These can be approximated with standard IEEE 754 floating-point numbers ("binary64" format, commonly known as "double precision"). Each such number is encoded over 64 bits, that split into the following elements:

- a sign $s = \pm 1$ (1 bit);
- an exponent e in the -1022 to $+1023$ range (11 bits);
- a mantissa m such that $1 \leq m < 2$ (52 bits).

In general, the represented value is $sm2^e$. The mantissa is encoded as $2^{52}(m-1)$; it has 53 bits of precision, but its top bit, of value 1 by definition, is omitted in the encoding.

The exponent e uses 11 bits, but its range covers only 2046 values, not 2048. The two extra possible values for that field encode special cases:

- The value zero. IEEE 754 has two zeros, that differ by the sign bit.
- Subnormals: they use the minimum value for the exponent (-1022) but the implicit top bit of the mantissa is 0 instead of 1.
- Infinites (positive and negative).
- Erroneous values, known as NaN (Not a Number).

Apart from zero, FALCON does not exercise these special cases; exponents remain relatively close to zero; no infinite or NaN is obtained.

The C language specification does not guarantee that its `double` type maps to IEEE 754 "binary64" type, only that it provides an exponent range and precision that match at least that IEEE type. Support of subnormals, infinites and NaNs is left as implementation-defined. In practice, most C compilers will provide what the underlying hardware directly implements, and *may* include full IEEE support for the special cases at the price of some non-negligible overhead, e.g. extra tests and supplementary code for subnormals, infinites and NaNs. Common x86 CPU, in 64-bit mode, use SSE2 registers and operations for floating-point, and the hardware already provides complete IEEE 754 support. Other processor types have only a partial support; e.g. many PowerPC cores meant for embedded systems do not handle subnormals (such values are then rounded to zeros). FALCON works properly with such limited floating-point types.

Some processors do not have a FPU at all. These will need to use some emulation using integer operations. As explained above, special cases need not be implemented.

3.3.2. FFT and NTT

3.3.2.1. FFT

The Fast Fourier Transform for a polynomial f computes $f(\zeta)$ for all roots ζ of ϕ (over $\mathbb{C}$). It is normally expressed recursively. If $\phi = x^n + 1$, and $f = f_0(x^2) + xf_1(x^2)$, then the following holds for any root ζ of ϕ:

$$\begin{array}{rcl} f(\zeta) & = & f_0(\zeta^2) + \zeta f_1(\zeta^2) \\ f(-\zeta) & = & f_0(\zeta^2) - \zeta f_1(\zeta^2) \end{array} \tag{3.36}$$

ζ^2 is a root of $x^{n/2} + 1$: thus, the FFT of f is easily computed, with $n/2$ multiplications and n additions or subtractions, from the FFT of f_0 and f_1, both being polynomials of degree less than $n/2$, and taken modulo $\phi' = x^{n/2} + 1$. This leads to a recursive algorithm of cost $O(n \log n)$ operations.

The FFT can be implemented iteratively, with minimal data movement and no extra buffer: in the equations above, the computed $f(\zeta)$ and $f(-\zeta)$ will replace $f_0(\zeta^2)$ and $f_1(\zeta^2)$. This leads to an implementation known as "bit reversal", due to the resulting ordering of the $f(\zeta)$: if $\zeta_j =$

$e^{i(\pi/2n)(2j+1)}$, then $f(\zeta_j)$ ends up in slot $\text{rev}(j)$, where rev is the bit-reversal function over $\log_2 n$ bits (it encodes its input in binary with left-to-right order, then reinterprets it back as an integer in right-to-left order).

In the iterative, bit-reversed FFT, the first step is computing the FFT of $n/2$ sub-polynomials of degree 1, corresponding to source index pairs $(0, n/2)$, $(1, n/2+1)$, and so on.

Some noteworthy points for FFT implementation in Falcon are the following:

- The FFT uses a table of pre-computed roots $\zeta_j = e^{i(\pi/2n)(2j+1)}$. The inverse FFT nominally requires, similarly, a table of inverses of these roots. However, $\zeta_j^{-1} = \overline{\zeta_j}$; thus, inverses can be efficiently recomputed by negating the imaginary part.

- ϕ has n distinct roots in $\mathbb{C}$, leading to n values $f(\zeta_j)$, each being a complex number, with a real and an imaginary part. Storage space requirements are then $2n$ floating-point numbers. However, if f is real, then, for every root ζ of ϕ, $\overline{\zeta}$ is also a root of ϕ, and $\overline{f(\zeta)} = f(\overline{\zeta})$. Thus, the FFT representation is redundant, and half of the values can be omitted, reducing storage space requirements to $n/2$ complex numbers, hence n floating-point values.

- The Hermitian adjoint of f is obtained in FFT representation by simply computing the conjugate of each $f(\zeta)$, i.e. negating the imaginary part. This means that when a polynomial is equal to its Hermitian adjoint (e.g. $ff^\star + gg^\star$), then its FFT representation contains only real values. If then multiplying or dividing by such a polynomial, the unnecessary multiplications by 0 can be optimized away.

- The C language (since 1999) offers direct support for complex numbers. However, it may be convenient to keep the real and imaginary parts separate, for values in FFT representation. If the real and imaginary parts are kept at indexes k and $k + n/2$, respectively, then some performance benefits are obtained:
  - The first step of FFT becomes free. That step involves gathering pairs of coefficients at indexes $(k, k + n/2)$, and assembling them with a root of $x^2 + 1$, which is i. The source coefficients are still real numbers, thus $(f_0, f_{n/2})$ yields $f_0 + if_{n/2}$, whose real and imaginary parts must be stored at indexes 0 and $n/2$ respectively, where they already are. The whole loop disappears.
  - When a polynomial is equal to its Hermitian adjoint, all its values in FFT representation are real. The imaginary parts are all null, and they represent the second half of the array. Storage requirements are then halved, without requiring any special reordering or move of values.

3.3.2.2. Ternary FFT

In the ternary case, the same general rules apply, but with some variations. Informally, in the binary case, the FFT is a succession of degree doublings; in the ternary case, there are three different operations:

- Initial step: modulus is $\phi = x^2 - x + 1$, whose roots are *not* i and $-i$; instead, its roots are $e^{i\pi/3}$ and $e^{-i\pi/3}$. The first step of the FFT is no longer free.

- Degree doublings are similar to the binary case; roots of $x^n - x^{n/2} + 1$ are the square roots of the roots of $x^{n/2} - x^{n/4} + 1$.

- Since n is a multiple of 3, there must be a degree tripling operation. Degree tripling uses the following equations:

$$\begin{array}{rcl} f(\zeta) & = & f_0(\zeta^3) + \zeta f_1(\zeta^3) + \zeta^2 f_2(\zeta^3) \\ f(\zeta\delta) & = & f_0(\zeta^3) + \zeta\delta f_1(\zeta^3) + \zeta^2\delta^2 f_2(\zeta^3) \\ f(\zeta\delta^2) & = & f_0(\zeta^3) + \zeta\delta^2 f_1(\zeta^3) + \zeta^2\delta f_2(\zeta^3) \end{array} \quad (3.37)$$

  where $\delta = e^i(2\pi/3)$ (a primitive cube root of 1).

The degree tripling operation can occur before the doublings, or after, or even in between. However, this choice must match the sequence of splittings and mergings in the Fast Fourier sampling. The description made in algorithm 3.38 elects to perform the degree tripling operation near the initial step, i.e. to bring the modulus from $x^2 - x + 1$ to $x^6 - x^3 + 1$.

3.3.2.3. NTT

The *Number Theoretic Transform* is the analog of the FFT, in the finite field $\mathbb{Z}_p$ of integers modulo a prime p. In the binary case, $\phi = x^n + 1$ will have roots in $\mathbb{Z}_p$ if and only if $p = 1 \bmod 2n$. In the ternary case, $\phi = x^n - x^{n/2} + 1$ is a divisor of $x^{3n/2} + 1$, hence we will need $p = 1 \bmod 3n$. The NTT, for an input polynomial f whose coefficients are integers modulo p, computes $f(\omega) \bmod p$ for all roots ω of ϕ in $\mathbb{Z}_p$.

Signature verification is naturally implemented modulo q. That small modulus was chosen precisely to allow the NTT to be used:

- Binary case: $q = 12289 = 1 + 12 \cdot 2048$
- Ternary case: $q = 18433 = 1 + 8 \cdot 2304$

Computations modulo q can be implemented with pure 32-bit integer arithmetics, avoiding divisions and branches, both being relatively expensive. For instance, modular addition of `x` and `y` may use this function:

```
static inline uint32_t
mq_add(uint32_t x, uint32_t y, uint32_t q)
{
    uint32_t d;

    d = x + y - q;
    return d + (q & -(d >> 31));
}
```

This code snippet uses the fact that C guarantees operations on `uint32_t` to be performed modulo 2^{32}; since operands fits on 15 bits, the top bit of the intermediate value `d` will be `1` if and only if the subtraction of `q` yields a negative value.

For multiplications, Montgomery multiplication is effective:

```
static inline uint32_t
mq_montymul(uint32_t x, uint32_t y, uint32_t q, uint32_t q0i)
{
    uint32_t z, w;
```

```
    z = x * y;
    w = ((z * q0i) & 0xFFFF) * q;
    z = ((z + w) >> 16) - q;
    return z + (q & -(z >> 31));
}
```

The parameter `q0i` contains $1/q \bmod 2^{16}$, a value which can be hardcoded since q is also known at compile-time. Montgomery multiplication, given x and y, computes $xy/(2^{16}) \bmod q$. The intermediate value `z` can be shown to be less than $2q$, which is why a single conditional subtraction is sufficient.

Modular divisions are not needed for signature verification, but they are handy for computing the public key h from f anf g, as part of key pair generation. Inversion of x modulo q can be computed in a number of ways; exponentation is straightforward: $1/x = x^{q-2} \bmod q$. For both 12289 and 18433, minimal addition chains on the exponent yield the result in 18 Montgomery multiplications (assuming input and output are in Montgomery representation).

Key pair generation may also use the NTT, modulo a number of small primes p_i, and the branchless implementation techniques described above. The choice of the size of such small moduli p_i depends on the abilities of the current architecture. The Falcon reference implementation, that aims at portability, uses moduli p_i which are slightly below 2^{31}, a choice which has some nice properties:

- Modular reductions after additions or subtractions can be computed with pure 32-bit unsigned arithmetics.
- Values may fit in the *signed* `int32_t` type.
- When doing Montgomery multiplications, intermediate values are less than 2^{63} and thus can be managed with the standard type `uint64_t`.

On a 64-bit machine with $64 \times 64 \to 128$ multiplications, 63-bit moduli would be a nice choice.

3.3.3. LDL Tree

From the private key properly said (the f, g, F and G short polynomials), signature generation involves two main steps: building the LDL tree, and then using it to sample a short vector. The LDL tree depends only on the private key, not the data to be signed, and is reusable for an arbitrary number of signatures; thus, it can be considered part of the private key. However, that tree is rather bulky (about 90 kB for $n = 1024$), and will use floating-point values, making its serialization complex to define in all generality. Therefore, the Falcon reference code rebuilds the LDL tree dynamically when the private key is loaded; its API still allows a built tree to be applied to many signature generation instances.

It would be possible to regenerate the LDL tree on the go, for a computational overhead similar to that of sampling the short vector itself; this would save space, since at no point would the full tree need to be present in RAM, only a path from the tree root to the current leaf. For degree n, a saved path would amount to about $2n$ floating-point values, i.e. roughly 16 kB. On the other hand, computational cost per signature would double.

Both LDL tree construction and sampling involve operations on polynomials, including multiplications (and divisions). It is highly recommended to use FFT representation, since multiplication and division of two degree-n polynomials in FFT representation requires only n elementary operations. The LDL tree is thus best kept in FFT.

3.3.4. Gaussian Sampler

When sampling a short vector, the inner Gaussian sampler is invoked twice for each leaf of the LDL tree. Each invocation should produce an integer value that follows a Gaussian distribution centered on a value μ and with standard deviation σ. The centers μ change from call to call, and are dynamically computed based on the message to sign, and the values returned by previous calls to the sampler. The values of σ are the leaves of the LDL tree: they depend on the private key, but not on the message; they range between 1.2 and 1.9 (in the ternary case, they may reach up to 2.20).

In the FALCON reference code, rejection sampling with regards to a bimodal Gaussian is used:

- The target μ is moved into the $[0..1[$ interval by adding an appropriate integer value, which will be subtracted from the sampling result at the end. For the rest of this description, we assume that $0 \leq \mu < 1$.
- A nonnegative integer z is randomly sampled following a half Gaussian distribution of standard deviation $\sigma_0 = 2$, centered on 0 (in the ternary case, we use $\sigma_0 = \sqrt{5} \approx 2.236$).
- A random bit b is obtained, to compute $z' = b + (2b - 1)z$. The integer z' follows a bimodal Gaussian distribution, and in the range of possible values for z' (depending on b), that distribution is above the target Gaussian of center μ and standard deviation σ.
- Rejection sampling is applied. z' follows the distribution:

$$G(z) = e^{-(z-b)^2/(2\sigma_0^2)} \tag{3.38}$$

  and we target the distribution:

$$S(z) = e^{-(z-\mu)^2/(2\sigma^2)} \tag{3.39}$$

  We thus generate a random bit d, whose value is 1 with probability:

$$\begin{aligned} P(d=1) &= S(z)/G(z) \\ &= e^{(z-b)^2/(2\sigma_0^2)-(z-\mu)^2/(2\sigma^2)} \end{aligned} \tag{3.40}$$

  If bit d is 1, then we return z'; otherwise, we start over.

Random values are obtained from a custom PRNG; the reference code uses ChaCha20, but any PRNG whose output is indistinguishable from random bits can be used. On a recent x86 CPU, it would make sense to use AES in CTR mode, to leverage the very good performance of the AES opcodes implemented by the CPU.

TODO

With a careful Rényi argument, the 53-bit precision of floating-point values used in the sampler computations are sufficient to achieve the required security levels.

3.3.5. Key Pair Generation

3.3.5.1. Gaussian Sampling

The f and g polynomials must be generated with an appropriate distribution. In the binary case, it is sufficient to generate each coefficient independently, with a Gaussian distribution centered on 0; values are easily tabulated.

In the ternary case, the coefficients should use a Gaussian distribution in the FFT embedding. For each of the $n/2$ coefficients $\hat{f}_j$ of the FFT representation of f, two uniformly random values a_j and b_j are generated in the $]0..1]$ range; the coefficient is then set to:

$$\hat{f}_j = \sigma\sqrt{-2\log a_j}e^{i2\pi b_j} \tag{3.41}$$

An inverse FFT is then applied, and the resulting values rounded to the nearest integers, to obtain f. The a_j and b_j needs not be generated with high precision; the FALCON reference code uses 32 random bits for each. The polynomial g is generated in a similar way.

3.3.5.2. Filtering

As per the FALCON specification, once f and g have been generated, some tests must be applied to determine their appropriateness:

- The $(g, -f)$ and its orthogonalized version must be short enough. In the ternary case, the norm must be measured on the FFT representation (this must be computed *after* the rounding of coefficients to integers, which may be done only in non-FFT representation).
- f must be invertible modulo ϕ and q; this is necessary in order to be able to compute the public key $h = g/f \bmod \phi \bmod q$. In practice, the NTT is used on f: all the resulting coefficients of f in NTT representation must be distinct from zero. Computing h is then straightforward.
- The FALCON reference implementation furthermore requires that $\mathrm{Res}(f, \phi)$ and $\mathrm{Res}(g, \phi)$ be both odd. If they are both even, the NTRU equation does not have a solution, but our implementation cannot tolerate that one is even and the other is odd. Computing the resultant modulo 2 is inexpensive; in the binary case, this is equal to the sum of the coefficients modulo 2.

If any of these tests fails, new (f, g) must be generated.

3.3.5.3. Solving The NTRU Equation

Solving the NTRU equation is formally a recursive process. At each depth:

1. Input polynomials f and g are received as input; they are modulo $\phi = x^n + 1$ for a given degree n.
2. New values $f' = \mathrm{N}(f)$ and $g' = \mathrm{N}(g)$ are computed; they live modulo $\phi' = x^{n/2} + 1$, i.e. half the degree of ϕ. However, their coefficients are typically twice longer than the coefficients of f and g.
3. The solver is invoked recursively over f' and g', and yields a solution (F', G') such that $f'G' - g'F' = q$.
4. Unreduced values (F, G) are generated, as:

$$\begin{array}{rcl} F & = & F'(x^2)g'(x^2)/g(x) \mod \phi \\ G & = & G'(x^2)f'(x^2)/f(x) \mod \phi \end{array} \tag{3.42}$$

   F and G are modulo ϕ (of degree n), and their coefficients have a size which is about three times that of the coefficients of inputs f and g.
5. Babai's nearest plane algorithm is applied, to bring coefficients of F and G down to that of the coefficients of f and g.

RNS and NTT. The operations implied in the recursion are much easier when operating on the NTT representation of polynomials. Indeed, if working modulo p, and ω is a root of $x^n + 1$ modulo p, then:

$$\begin{aligned} f'(\omega^2) &= N(f)(\omega^2) = f(\omega)f(-\omega) \\ F(\omega) &= F'(\omega^2)g(-\omega) \end{aligned} \tag{3.43}$$

Therefore, the NTT representations of f' and g' can be easily computed from the NTT representations of f and g; and, similarly, the NTT representation of F and G (unreduced) are as easily obtained from the NTT representations of F' and G'. This naturally leads to the use of a Residue Number System (RNS), in which a value x is encoded as a sequence of values $x_j = x \bmod p_j$ for a number of distinct small primes p_j. In the FALCON reference implementation, the p_j are chosen such that $p_j < 2^{31}$ (to make computations easy with pure integer arithmetics) and $p_j = 1 \bmod 2048$ (to allow the NTT to be applied). Conversion from the RNS encoding to a plain integer in base 2^{31} is a straightforward application of the Chinese Remainder Theorem; if done prime by prime, then the only required big-integer primitives will be additions, subtractions, and multiplication by a one-word value. In general, coefficient values are signed, while the CRT yields values ranging from 0 to $\prod p_j - 1$; normalisation is applied by assuming that the final value is substantially smaller, in absolute value, than the product of the used primes p_j.

Coefficient sizes. Key pair generation has the unique feature that it is allowed occasional failures: it may reject some cases which are nominally valid, but do not match some assumptions. This does not induce any weakness or substantial performance degradation, as long as such rejections are rare enough not to substantially reduce the space of generated private keys. In that sense, it is convenient to use *a priori* estimates of coefficient sizes, to perform the relevant memory allocations and decide how many small primes p_j are required for the RNS representation of any integer at any point of the algorithm. The following maximum sizes of coefficients, in bits, have been measured over thousands of random key pairs, at various depths of the recursion (in the binary case):

depth	**max** f, g	**std. dev.**	**max** F, G	**std. dev.**
10	6307.52	24.48	6319.66	24.51
9	3138.35	12.25	9403.29	27.55
8	1576.87	7.49	4703.30	14.77
7	794.17	4.98	2361.84	9.31
6	400.67	3.10	1188.68	6.04
5	202.22	1.87	599.81	3.87
4	101.62	1.02	303.49	2.38
3	50.37	0.53	153.65	1.39
2	24.07	0.25	78.20	0.73
1	10.99	0.08	39.82	0.41
0	4.00	0.00	19.61	0.49

These sizes are expressed in bits; for each depth, each category of value, and each key pair, the maximum size of the absolute value is gathered. The array above lists the observed averages and standard deviations for these values. A FALCON key pair generator may thus simply assume that values fit correspondingly dimensioned buffers, e.g. by using the measured average added to, say, six times the standard deviation. This would ensure that values almost always fit. A final test at the end of the process, to verify that the computed F and G match the NTRU equation, is sufficient to detect failures. Note that for depth 10, the maximum size of F and G is the one resulting from the extended GCD, thus similar to that of f and g.

Binary GCD. At the deepest recursion level, inputs f and g are plain integers (the modulus is $\phi = x + 1$); a solution can be computed directly with the Extended Euclidean Algorithm, or a variant thereof. The FALCON reference implementation uses the binary GCD. This algorithm can be expressed in the following way:

- Values a, b, u_0, u_1, v_0 and v_1 are initialized and maintained with the following invariants:

$$\begin{aligned} a &= fu_0 - gv_0 \\ b &= fu_1 - gv_1 \end{aligned} \tag{3.44}$$

  Initial values are:

$$\begin{aligned} a &= f \\ u_0 &= 1 \\ v_0 &= 0 \\ b &= g \\ u_1 &= g \\ v_1 &= f - 1 \end{aligned} \tag{3.45}$$

- At each step, a or b is reduced: if a and/or b is even, then it is divided by 2; otherwise, if both values are odd, then the smaller of the two is subtracted from the larger, and the result, now even, is divided by 2. Corresponding operations are applied on u_0, v_0, u_1 and v_1 to maintain the invariants. Note that computations on u_0 and u_1 are done modulo g, while computations on v_0 and v_1 are done modulo f.
- Algorithm stops when $a = b$, at which point the common value is the GCD of f and g.

If the GCD is 1, then a solution $(F, G) = (qv_0, qu_0)$ can be returned. Otherwise, the FALCON reference implementation rejects the (f, g) pair. Note that the (rare) case of a GCD equal to q itself is also rejected; as noted above, this does not induce any particular algorithm weakness. The description above is a bit-by-bit algorithm. However, it can be seen that most of the decisions are taken only on the low bits and high bits of a and b. It is thus possible to group updates of a, b and other values by groups of, say, 31 bits, yielding much better performance.

Iterative version. Each recursion depth involves receiving (f, g) from the upper level, and saving them for the duration of the recursive call. Since degrees are halved and coefficients double in size at each level, the storage space for such an (f, g) pair is mostly constant, around 13000 bits per depth. For $n = 1024$, depth goes to 10, inducing a space requirement of at least 130000 bits, or 16 kB, just for that storage. In order to reduce space requirements, the FALCON reference implementation recomputes (f, g) dynamically from start when needed. Measures indicate a relatively low CPU overhead (about 15%). A side-effect of this recomputation is that each recursion level has nothing to save. The algorithm thus becomes iterative.

Babai's reduction. When candidates F and G have been assembled, they must be reduced against the current f and g. Reduction is performed as successive approximate reductions, that are computed with the FFT:

- Coefficients of f, g, F and G are converted to floating-point values, yielding $\dot{f}$, $\dot{g}$, $\dot{F}$ and $\dot{G}$. Scaling is applied so that the maximum coefficient of $\dot{F}$ and $\dot{G}$ is about 2^{30} times the maximum coefficient of $\dot{f}$ and $\dot{g}$; scaling also ensures that all values fit in the exponent range of floating-point values.

- An integer polynomial k is computed as:

$$k = \left\lfloor \frac{\dot{F}\dot{f}^\star + \dot{G}\dot{g}^\star}{\dot{f}\dot{f}^\star + \dot{g}\dot{g}^\star} \right\rceil \tag{3.46}$$

  This computation is typically performed in FFT representation, where multiplication and division of polynomials are easy. Rounding to integers, though, must be done in coefficient representation.

- kf and kg are subtracted from F and G, respectively. Note that this operation must be exact, and is performed on the integer values, not the floating-point approximations. At high degree (i.e. low recursion depth), RNS and NTT are used: the more efficient multiplications in NTT offset the extra cost for converting values to RNS and back.

This process reduces the maximum sizes of coefficients of F and G by about 30 bits at each iteration; it is applied repeatedly as long as it works, i.e. the maximum size is indeed reduced. A failure is reported if the final maximum size of F and G coefficients does not fit the target size, i.e. the size of the buffers allocated for these values.

3.3.6. Performances

The Falcon reference implementation achieves the following performance on an Intel® Core® i7-6567U CPU (clocked at 3.3 GHz):

degree	keygen (ms)	keygen (RAM)	sign/s	vrfy/s	pub length	sig length
512	6.98	14336	6081.9	37175.3	897	617.38
768	12.69	27648	3547.9	20637.7	1441	993.91
1024	19.64	28672	3072.5	17697.4	1793	1233.29

The following notes apply:

- RAM usage for key pair generation is expressed in bytes. It includes temporary buffers for all intermediate values, including the floating-point polynomials used for Babai's reduction.

- Public key length and average signature length are expressed in bytes. The size of public keys includes a one-byte header that identifies the degree and modulus. For signatures, compression is used, which makes the size slightly variable; the average is reported here.

- The Falcon reference implementation uses only standard C code, not inline assembly, intrinsics or 128-bit integers. In particular, it is expected that replacing the internal PRNG (a straightforward, portable implementation of ChaCha20) with AES-CTR using the dedicated CPU opcodes will yield a substantial performance improvement. SSE2 and AVX opcodes should help with FFT, and 64-bit multiplications (with 128-bit results) might improve key generation time as well.

- The i7-6567U processor implements dynamic frequency scaling based on load and temperature. As such, measures are not very precise and tend to move by as much as 15% between any two benchmark runs.

- Signature generation time does not include the LDL tree building, which is done when the private key is loaded. These figures thus correspond to batch usage, when many values must be signed with a given key. This matches, for instance, the use case of a busy TLS server. If, in a specific scenario, keys are used only once, then the LDL tree building cost must be added to each signature attempt. It is expected that this would about double the CPU cost of each signature.

Far better it is to dare mighty things, to win glorious triumphs, even though checkered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the gray twilight that knows not victory nor defeat.

The Strenuous Life – Theodore Roosevelt

Delegating Elliptic-Curve Operations with Homomorphic Encryption

A

This chapter is a detailed version of the article *Delegating Elliptic-Curve Operations with Homomorphic Encryption* coauthored with Carlos Aguilar-Melchor, Jean-Christophe Deneuville, Philippe Gaborit, Tancrède Lepoint and published at SPC 2018. It is organized as follows: Sec. A.1 introduces the general approach to scalar multiplication delegation, and describes our approach. Experimental results for our protocol are presented in Sec. A.2.

Contents

Regarded as cryptography's "Holy Grail", fully homomorphic encryption (FHE) enables computation of (almost) arbitrary functions on encrypted data [Gen09; BGV12]. Secure delegation of computationally costly operations is one of the most intuitive applications of homomorphic encryption. Indeed, instead of computing a given function for some secret inputs the delegator can encrypt the inputs, send them to a delegatee that computes the function over the encrypted data using FHE, and finally the delegator can retrieve the result and decrypt it.

The usual approach to compute a function through FHE is to find a binary circuit with AND and XOR gates allowing to evaluate the function. As a XOR gate is equivalent to an addition modulo two and AND gate to a multiplication modulo two, if an FHE scheme allows to do enough plaintext additions and multiplications modulo two, it can be used to evaluate this circuit over bit-by-bit encrypted data.

In practice, the computational cost of such an approach, both for the delegator and the delegatee, grows rapidly with the multiplicative-depth of the FHE computation, which is equal to the AND-depth (i.e. the maximal amount of successive AND gates an input must go through) of the circuit. Therefore, it is important to obtain low AND-depth circuits for the functions that must be evaluated.

Another interesting class of circuits are *arithmetic circuits*. In the context of FHE, these circuits are defined as circuits in which gates are exclusively integer additions or multiplications modulo a given fixed modulus (no division, inversions, bit operations, floating-point representation, etc.). When the function to be evaluated is a modular arithmetic polynomial, we can reduce significantly the multiplicative depth by replacing the binary circuit with an arithmetic circuit. In this case, we can directly use an FHE's natural addition and multiplication to evaluate the circuit by setting its plaintext modulus as the circuit modulus.

Point computation. The considered points on elliptic-curve cryptography belong to a cyclic group, and therefore every point can be obtained through a scalar multiplication with a given generator. This is generally the most costly operation that is required on a protocol based on elliptic-curve cryptography. Hence, as in previous works, we focus on delegating this operation.

Computing a scalar multiplication can be reduced, through a classical double-and-add [Knu97] algorithm, to computing additions of points. Point addition algorithms depend a lot on the coordinate system used. In this sense, the most popular are projective coordinates which allow to make scalar multiplication of elliptic-curve points with a shallow arithmetic circuit. Combining these two facts, we deduce that it is possible to compute a scalar multiplication in an elliptic-curve through shallow arithmetic circuits.

This work. Practical implementations of FHE are not well suited for large plaintext moduli which are essential in elliptic-curve cryptography. In order to evaluate the efficiency of homomorphic encryption in this setting we have modified some existing libraries and proposed some new homomorphic encryption implementations that are adapted to it. We expect those libraries, to be released with an open-source license, to be useful in other contexts too.

Our delegation protocol is generic and it is possible to delegate the computation of a scalar operation over generic groups. Thus it is also possible to use the delegation techniques described here over other groups such as RSA multiplicative groups, but even with optimizations such as RNS representation *and* Montgomery Reductions (such as [BI04]), our tests demonstrate that the computational cost is too high to consider practical benefits in a foreseeable future. Given this fact, and space constraints, we focus on the groups used on elliptic-curve cryptography. More precisely we will use as an example computations over P-256 [NIS16] which grants 128 bits security for

cryptographic protocols.

In order to provide an efficient delegation protocol in the elliptic-curve setting we have revisited the literature on elliptic-curve point additions. Usually these algorithms try to reduce the number of field operations done, whereas in our setting we want to have the lowest possible multiplicative depth. We show that a recent algorithm, by Renes *et al.* [**cryptoeprint:2015:1060**] gives an algorithm that fits perfectly the homomorphic encryption setting with only two levels of multiplications.

A straightforward delegation protocol of the double-and-add protocol would result in a circuit with a very large multiplicative depth. For example for P-256 we would have 512 point addition depth (to hide through 256 iterations whether we do an add or a double-and-add). Using the point addition described above this will result in a multiplicative depth of 1024 which is absolutely unreachable with current homomorphic encryption schemes.

We propose an improved protocol, which relies on pre-computation and windowing, leading to delegation protocols such that the arithmetic circuit to be evaluated is extremely shallow. The exact depth depends on the pre-computation memory / performance trade-off. For example for a client with 64 MBytes of pre-computed data the delegatee only needs to evaluate an arithmetic circuit with multiplicative depth 8 in the P-256 setting.

The main bottleneck for our delegation protocol is the amount of data that needs to be sent to the delegatee. FHE encryption has a low transmission rate and encrypting and sending large amounts of data is therefore the most costly operation for the client. Again for the example of P-256 with 64MBytes of pre-computed data, the client has to send 200Kbits for every scalar multiplication done. Thus even with a gigabit Ethernet connection he cannot delegate more than five thousand multiplications per second. From a computational point of view the protocol is realistic and requires a few milliseconds of computation from the server per scalar multiplication.

Note that if we reduce the computational constraints on the delegatee and reduce to the maximum the delegator costs we can use a trans-encryption technique [MJSC16]. With such an approach the client only needs to send 16Kbits per scalar multiplication and can delegate a much larger amount of operations.

The practicality of our technique is validated through proof-of-concept implementations of the delegator and delegatee. Our implementations are based on an adaptation of HElib [HS14; HS15] so as to handle large plaintext spaces (HElib is restricted to single precision prime powers). The delegatee implementation has been deployed on an Amazon EC2 instance (`c4.8xlarge`) to allow reproducibility and verification of our results.

HE practical implementation. The most popular library, HELib [HS14; HS15] was developed by Halevi and Shoup. It relies on Shoup's Number Theory Library [Sho15] (NTL) and implements a variant of the BGV [BGV12] scheme of Brakerski, Gentry and Vaikuntanathan. It also features many optimizations proposed by Gentry, Halevi and Smart [GHS12c]. HElib restricts the modulus to $t = p^r$ with p and r being simple-precision integers. We modified this library to accept such moduli but without size restrictions on p.

Precomputations and delegated computations. Many works in the early 90's proposed to speed up protocols either based on factoring [Sch91; BGMW93; LL94; de 95; BPV98] or discrete logs [BPV98] by using precomputations. While we also use precomputations in our protocol, most of speed-up is achieved through the delegation of some computations to an external, While we also use precomputations in our protocol (see Fig. A.1 for an overview and A.3 for more details), most of speed-up is achieved through the delegation of some computations to an external, untrusted source

of computational power, using additional techniques such as windowing [Knu97].

Additionally, several protocols to delegate computation have been proposed in the literature, especially regarding to the delegation of scalar operations [Fei86; MKI90; NS01; HL05; VCG+06; CLV16]. In [CLV16], Chevalier *et al.* present various delegation protocols and a proof that the attained performance is optimal. However, the proof they give considers that the only operation that the delegatee can do is a set of scalar multiplications for scalars given in the clear. Using homomorphic encryption we can make the delegatee compute a scalar multiplication without knowing which scalar is being used and therefore we can go beyond their definition of optimality.

Prototypes using homomorphic encryption. Open-source libraries for homomorphic encryption are available online [LN14; HS14; HS15; MBG+16], and were used as building blocks in several prototypes in the recent years, such as statistics computations [NLV01], machine learning [GLN13], signal processing [AFF+13], database queries [BGH+13; CKK15], private health diagnosis [BLN14], genome statistics [LLN15], and edit distance [CKL15]. To the best of our knowledge, no prototype tackled the delegation of scalar multiplications in the elliptic curve setting (nor exponentiations in the RSA setting).

Notations. Let $\mathbb{Z}_q$ be the set of integers modulo q, $\mathbb{F}_q$ be the finite field of q elements, and denote $E(\mathbb{F}_q)$ the group of points $(x, y) \in \mathbb{F}_q^2$ belonging to an elliptic curve E and G an element of $E(\mathbb{F}_q)$ (which in practice will often be a generator). Sampling uniformly an element x from a set S is denoted $x \xleftarrow{\$} S$. If q is an integer, denote $\ell_{w,q}$ the length of its representation using a basis of w elements (*e.g.* $\ell_{2,q}$ is its bit length).

ECDSA. We briefly recall the ECDSA signature scheme. Introduced by Vanstone in 1992 (see [HMV03]), ECDSA is the elliptic-curve analogue of DSA and features short keys and signature sizes. Its signature and verification procedures are provided in algorithms 1.42 and 1.43. We do not recall the key generation procedure, which is of no relevance to this work. The domain parameters provided

Algorithm 1.42 ECDSA.Sign

Require: $m, pk = (q, E, G, Q = sG, n)$ and $sk = s$, m being a message, q an integer, E an elliptic curve over $\mathbb{F}_q$, G, Q two points of E, s an element of $\mathbb{F}_q$, and n the order of G on E
Ensure: ECDSA signature $(r, t) \in \mathbb{F}_q^2$ of m

1: **repeat**
2: $k \xleftarrow{\$} \{1, \ldots, n-1\}$
3: $(x, y) \leftarrow kG$
4: $r \leftarrow x \bmod n$
5: $t \leftarrow k^{-1}(h(m) + sx) \bmod n$
6: **until** $r \neq 0$ **and** $t \neq 0$
7: **return** (r, t)

by the NIST include three types of curves: over prime fields (P-xxx), over binary fields (B-xxx) or Koblitz curves (K-xxx). Note that, when compared to prime curves, curves over binary fields and Koblitz curves would reduce considerably the outsourcing costs given their small characteristic. However, such curves are not usually proposed in the client setting (*e.g.* in the implementations we compare to). For this reason, in this work, we only consider elliptic curves over prime fields.

Algorithm 1.43 ECDSA.Verif

Require: q, E, G, Q, (r,t), m, as defined in **ECDSA.Sign**
Ensure: **true** if and only if (r,t) is a valid signature on m

```
if Q = ∞ or Q ∉ E(F_q) then
    return false
else
    (x, y) ← (h(m)t^{-1} mod n)G + (rt^{-1} mod n)Q
    if r = x mod n then
        return true
    else
        return false
```

For such curves, q denotes the field size, (a,b) are two field elements defining the short Weierstraß equation

$$y^2 = x^3 + ax - b, \text{ with } 4a^3 + 27b^2 \neq 0 \mod q,$$

and G is a base point of prime order n on the curve. As usual for such curves, we consider that $|E(\mathbb{F}_q)| = n$ (*i.e.* a cofactor $h = 1$).

Homomorphic encryption and HElib. An homomorphic encryption (HE) scheme [Gen09] allows to publicly process encrypted data without knowing the secret key. In this section we recall the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [BGV12] — implemented in the software library HElib [HS14; HS15] — that we use in our prototypes. This description is mostly taken from [GHS12c].

BGV. BGV is defined over polynomial rings of the form $R = \mathbb{Z}[x]/(\Phi_m(x))$ where m is a parameter and Φ_m the m-th cyclotomic polynomial. The plaintext space is usually the ring $R_p = R/pR$ for an integer p.

A plaintext polynomial $a(x) \in R_p$ is encrypted as a vector over $R_q = R/qR$, where q is an odd public modulus. More specifically, BGV contains a chain of moduli of decreasing size $q_0 > q_1 > \cdots > q_L$ and freshly encrypted ciphertexts are defined modulo q_0. During homomorphic evaluation, we keep switching to smaller moduli after each multiplication until we get ciphertexts modulo q_L, which cannot be multiplied anymore — L is therefore an upper bound on the multiplicative depth of the circuit we can compute.

Homomorphic operations. The plaintext space of BGV are elements of R_p, and homomorphic additions (resp. multiplications) correspond to additions (resp. multiplications) over the ring R_p.[1]

Batching. Rather than encrypting elements of $R_p = \mathbb{Z}_p[x]/(\Phi_m(x))$, the BGV cryptosystem can be slightly modified to encrypt vectors of elements of $\mathbb{Z}_p$ [SV10; SV14; CCK+13] in an SIMD fashion: homomorphic operations implicitly perform the componentwise operations over the plaintext vectors (and rotations are easy to perform via the Frobenius endomorphism). Such a feature is called *batching* and essentially allows, for the cost of an homomorphic evaluation, to evaluate the function independently on several inputs [BGV12]. Let us recall briefly the batching technique for

[1] One can easily obtain a scheme with plaintext space $\mathbb{Z}_p$ by embedding $\mathbb{Z}_p$ into R_p via $a \in \mathbb{Z}_p \mapsto a \in R_p$. Hence homomorphic operations correspond to arithmetic operations over the ring $\mathbb{Z}_p$.

BGV from [SV10; SV14]. If the cyclotomic polynomial $\Phi_m(x)$ factors modulo the plaintext space p into a product of irreducible factors $\Phi_m(x) = \prod_{j=0}^{\ell-1} F_j(x) \pmod p$, then a plaintext polynomial $a(x) \in R_p$ can be viewed as encoding ℓ different small polynomials, $a_j = a \bmod F_j$, and each constant coefficient of the a_j can be set to an element of $\mathbb{Z}_p$. Unfortunately, not every tuple (p, m) yield an efficient batching — we will discuss how we selected (p, m) in Sec. A.2.1.

HElib. HElib [HS14; HS15] is, as of today, a standard library for HE prototypes. HElib is a C++ library that implements the BGV scheme (with batching) for arbitrary plaintext space modulus p^r, with p and r in simple-precision. This software library uses NTL [Sho15] and includes numerous optimizations described in [GHS12c; HS14; HS15]. HElib supports multi-threading and is distributed under the terms of the GNU GPL version 2.

A.1. Delegating Elliptic-Curve Point Computation

Recall that we want to delegate the computation of kG to an Untrusted Cloud (UC) for a secret k and a public or secret G. The high-level idea of the protocol is that we will send k and G (actually a representation thereof) encrypted under an HE scheme $\mathcal{E}$ so that the Cloud can homomorphically compute kG without knowing k nor G. Then the client will be able to decrypt and obtain kG. Note that our protocol, described in this section, does not reveal G but this point can be publicly known by other means (e.g. if using a standard which specifies a generator that should be used in a protocol). Assume the client C is connected to an external source of computational power (*e.g.* the Cloud, a LAN server or a Cryptographic Coprocessor in a PCI-e card) that we denote UC. In Sec. A.1.2, we give a high-level overview of our protocol; in the rest of the section, we detail each step of the protocol, and in Sec. A.1.6 we give the full protocol for completeness.

A.1.1. Throughput Limitation

Potentially, the computational power of the UC could be as large as needed. In such a setting the bottleneck can be two-fold: (1) the client communication bandwidth and (2) the HE scheme encryption/decryption throughput and the client post-processing. In practice, both potential bottlenecks have a similar impact, and give an upper bound on the amount of encrypted data that can enter/exit the client.

If the UC is in a LAN, or even better in a PCI-e card, available bandwidths can be very high. In the PCI-e setting, bandwidth can theoretically reach 252.064 Gbit/s (PCI-e x32 v4.0 with 16-lane slot). However, the client will not be able to do homomorphic encryptions or decryptions at such speeds. In practice a throughput limitation around some Gigabits will always exist, and will be the main performance bottleneck.

A.1.2. High-Level Description

The high-level description of this delegation protocol is provided in figure A.1. In a first step, some precomputation will be performed on G (see Sec. A.1.3 — essentially it will compute a set of windowed scalar multiplications of G), then in Step 2), some of the precomputed values, useful to compute kG, are encrypted under the HE scheme $\mathcal{E}$ and sent to the UC. The UC will homomorphically compute kG in Step 3), and will send back the computation to the client (Step 4)). Finally, the client will decrypt the result.

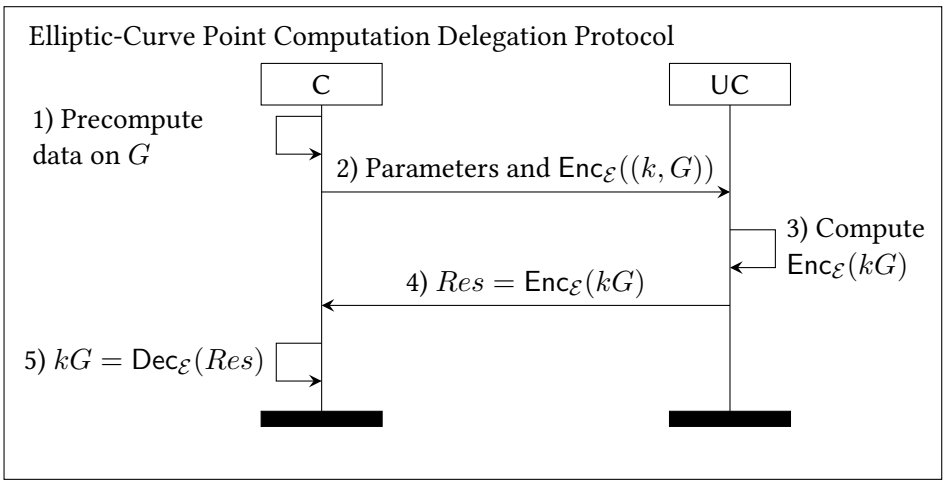


Figure A.1. – High-level description of the delegation protocol: delegation of the elliptic-curve point computation.

In the rest of this section, we will specify in detail every step of this protocol, and namely: (i) what precomputations need to be performed on G (Sec. A.1.3), (ii) which server computation, EC representation, and point addition should be used (Sec. A.1.5).

A.1.3. Step 1): Initialization and Windowing

Each doubling or point addition to compute kG from k will increase the multiplicative depth of the UC computation. Therefore, we use a classical window technique to trade computation complexity for memory.[2] In Sec. A.1.5, we will detail how to represent the EC points to obtain an addition formula of multiplicative depth 2.

More precisely during step 1, for a ω bits window size, the client precomputes all $2^{\omega} \cdot \ell_{2,n}/\omega$ points of the form $P_{i,j} = (i \cdot 2^{\omega(j-1)})G$ for i from 0 to $2^{\omega} - 1$, and for j from 1 to $\ell_{2,n}/\omega$ (we discuss the memory implications of this windowing technique in Sec. A.2.2). Thus, for every scalar $k = \sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} \cdot 2^{\omega j}$, we have that

$$kG = \left(\sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} 2^{\omega j} \right) G = \sum_{j=1}^{\ell_{2,n}/\omega} P_{k^{(j)},j} \, ,$$

and computing the scalar multiplication only costs $\ell_{2,n}/\omega$ point additions.

The whole process is described in figure A.3 for a single delegated EC point computation. Note that the main cost for the client is to send the ordered set $(\mathsf{Enc}_{\mathcal{E}}(x_j), \mathsf{Enc}_{\mathcal{E}}(y_j))_j$ for $P_{k_j,j} = (x_j, y_j)$ and $j \in [0..\ell_{2,n}/\omega)$.

A.1.4. Using HE Batching

Instead of encrypting only one x_j (resp. y_j) per ciphertext, we will use HE batching to encrypt several (say m of them) x_j's in parallel for *different inputs scalars* k's (and the same j). Thus, for the

[2] We leave as an interesting open problem a fine-grained analysis of the homomorphic evaluations of the best time-memory trade-offs for regular implementation of scalar multiplication over prime-field elliptic curves [Riv11].

same communication and UC computation complexities, we will be able to compute several kG's in parallel.

A.1.5. Steps 2) and 3): Revisiting Elliptic Curve Point Additions

In EC cryptography there are dozens of ways to perform additions and point doubling (see [HMV03] for instance). Most algorithms were designed to minimize the number of multiplications on $\mathbb{F}_q$ one has to perform.[3] In practice the existing algorithms are not a priori optimized in terms of multiplicative circuit depth and we must revisit the existing work to get the best possible algorithm given this new optimization target.

In the rest of this chapter, we focus on ECs with short Weierstraß equation of the form $E(\mathbb{F}_q)$: $y^2 = x^3 - 3x + b$, for $q \geq 5$ (the ones used in NIST standards). Recently Renes *et al.* proposed an efficient complete addition law that is valid not only for curves of composite order, but also for all prime order NIST curves [RCB16], using standard projective coordinates. In such a representation, points $P(x, y) \in E(\mathbb{F}_q)$ can be written with triple (X, Y, Z) where $x = X/Z$ and $y = Y/Z$ [HMV03]. Such coordinates were designed to speed up the point addition and doubling computation by avoiding field inversions for the benefit of multiplications.

Renes *et al.* presented an addition law [RCB16, Algorithm 4] point addition circuits with a multiplicative depth of 2; we briefly review these formulae. Let $P_1(X_1, Y_1, Z_1)$ and $P_2(X_2, Y_2, Z_2)$ be points in the projective embedding of $E(\mathbb{F}_q)$, and denote by $P_3(X_3, Y_3, Z_3)$ their sum. Notice that there are no requirements on P_1 and P_2 being different, nor on being distinct from $\mathcal{O}(0, 1, 0)$.

For sake of clarity, we introduce some intermediate variables that can be computed with a multiplicative depth of one:

$$
\begin{aligned}
T_0 &= (X_1Y_2 + X_2Y_1), & T_3 &= Y_1Z_2 + Y_2Z_1,\\
T_1 &= Y_1Y_2, & T_4 &= b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2,\\
T_2 &= 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2), & T_5 &= 3(X_1X_2 - Z_1Z_2).
\end{aligned}
$$

Then the complete addition law on the projective embedding is given by the formula:

$$
\begin{aligned}
X_3 &= T_0(T_1 + T_2) - 3T_3T_4,\\
Y_3 &= (T_1 - T_2)(T_1 + T_2) + 3T_5T_4,\\
Z_3 &= T_3(T_1 - T_2) + T_0T_5.
\end{aligned}
$$

As mentioned in [RCB16], the "plaintext" cost of this formula is $12\mathbf{M} + 2\mathbf{m}_b + 29\mathbf{a}$, where $\mathbf{M}$ (resp. $\mathbf{m}_b$ resp. $\mathbf{a}$) is the cost of a multiplication of two field elements (resp. multiplication by a constant, resp. addition). Therefore, the cost of evaluating this formula in the encrypted domain is 12 homomorphic multiplications, plus 2 multiplications by a constant, plus 29 homomorphic additions. The main advantage of this formula is that it can be evaluated as a depth 2 circuit.

On a different but non negligible side, this addition law is also optimal in terms of communication complexity. Indeed, as it requires at least three coordinates to avoid field inversions — which are

[3] Also they were designed to resist side-channel attacks by using the same "unified" formula for addition and point doubling, but we do not have to worry about such attacks because in homomorphic encryption, the operation flow is independent of the input scalar.

problematic for homomorphic computations — the formula of Renes *et al.* reaches the optimal lower bound. Moreover, the points we start our additions with are on standard coordinates which means that when putting them into projective coordinates, the third coordinate is always 1. We can therefore send just two coordinates per point.

For our proof-of-concept implementation of the protocol using a modified HElib BGV, [RCB16, Algorithm 4] turned out to be the addition law yielding best performances. This can be explained by the relatively low number of multiplications required by this formula, additionally to the optimal depth and representation.

A reasonable question is whether one can reduce significantly the amount of operations by using more coordinates or increasing depth. The short answer proved to be no given the literature on EC point addition formulas (see [BL08]). Roughly, a deeper circuit can sometimes save one coordinate in the point representation, but the FHE parameters become too large. Vice-versa an additional coordinate either imply additional multiplications or communications that are not worth it. Nevertheless, tradeoffs for constrained devices are possible and finding the best tuple (EC, set of coordinates, addition algorithm, homomorphic encryption scheme) for computing over encrypted EC points remains an interesting open question.

Figure A.2 below shows that the complete addition formula of [**cryptoeprint:2015:1060**] has multiplicative depth 2.

A.1.6. Full Protocol

For completeness, we provide in figure A.3 the full description of the protocol. For the delegator C, the main effort is on steps 5 and 6. They define his maximum capacity to delegate the point computations.

A.1.7. Security

As usual in delegation protocols we consider the delegatee honest but curious. In this setting the IND-CPA property of our scheme combined with a standard hybrid argument ensures the attacker learns nothing about the plaintexts sent to him nor about the result of the computation.

In this section, we provide the basic definitions of security of the underlying schemes, and show that in particular the protocol is secure against a semi-honest adversary when used to delegate the ECDSA signature scheme. We then discuss the malicious adversary setting.

A.1.7.1. Basic definitions

The usual definition of security for an homomorphic encryption scheme is indistinguishability against chosen plaintext attacks. The encryption scheme we use ensures this property under standard assumptions [BGV12].

Definition A.1 (IND-CPA). *Let $\mathcal{E} = (KeyGen, Enc, Dec)$ be a encryption scheme, and let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary. $\mathcal{E}$ is indistinguishable under chosen-plaintext attack (IND-CPA) if for all pair of plaintexts (m_0, m_1), we have*

$$\mathrm{Adv}^{\mathcal{A}} = |\Pr[\mathcal{A}(Enc(m_0)) = 1] - \Pr[\mathcal{A}(Enc(m_1)) = 1]|$$

is negligible.

For signature schemes, the standard definition of security is (existential) unforgeability against chosen message attacks. ECDSA ensures this property in the random oracle model [Vau03].

Definition A.2 (UF-CMA). *Let $\mathcal{S} = (KeyGen, Sign, Verif)$ be a signature scheme, and let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary which can sign messages of his choice to produce any message properly signed, with the exception of this message itself. $\mathcal{S}$ is unforgeable under chosen messages attack (UF-CMA) if we have*

$$\mathrm{Adv}^{\mathcal{A}} = \Pr\left[\mathcal{A}^{Sign(sk,\cdot)}(pk) = (m,\sigma); Verif(pk,m,\sigma) = 1\right]$$

is negligible.

A.1.7.2. Proof of Security for the Outsourcing of ECDSA

The main goal of this section is to show that the outsourcing protocol does not result in loosing the UF-CMA property when considering the new inputs an attacker may have. In other words we want to show that if the ECDSA is UF-CMA and the encryption scheme is IND-CPA, then the resulting outsourced protocol is also UF-CMA.

The basic idea in this proof is that an adversary able to use the encrypted output with the IND-CPA scheme to break the UF-CMA of the global scheme is also able to break the security of the IND-CPA scheme or able to break the UF-CMA of the ECDSA protocol. Indeed if he is not able to break the UF-CMA of the ECDSA protocol, then he can build a distinguisher for an IND-CPA challenge.

For clarity, the notation related to coordinates has been removed, and a simplified description of the protocol is considered.

Algorithm 1.44 $\mathrm{OP}_{\mathrm{ECDSA}}$

Require: $(h(m_i))_{1\leq i\leq b}$
Ensure: $(\sigma_i)_{1\leq i\leq b} = (r_i, t_i)_{1\leq i\leq b}$

1: C generates $k_i \xleftarrow{\$} \{1,\ldots,n-1\}$ in basis 2^ω
2: C sends $\left(\mathrm{Enc}_{\mathcal{E}}\left(\left(P_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)\right)_{1\leq j\leq \ell}$ to UC
3: UC sends $\sum_{j=1}^{\ell}\mathrm{Enc}_{\mathcal{E}}\left(\left(P_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)$ to C
4: C returns $(r_i = x_i \bmod n, t_i = k_i^{-1}(h(m_i)+s\cdot x_i) \bmod n)_{1\leq i\leq b}$ where $(x_i,y_i) = \sum_j P_{k_i^{(j)},j} = k_iG$

Theorem A.1. *If $\mathcal{E}$ is IND-CPA and ECDSA is UF-CMA, then OP_{ECDSA} is UF-CMA.*

Proof. To prove security we use the following games.

Game 0: This is $\mathrm{OP}_{\mathrm{ECDSA}}$ itself;

Game 1: This game is the same as **Game 0**, except that the client computes k_iG by itself in step 4.

1. C generates $k_i \xleftarrow{\$} \{1,\ldots,n-1\}$ in basis 2^ω
2. C sends $\left(\mathrm{Enc}_{\mathcal{E}}\left(\left(P_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)\right)_{1\leq j\leq \ell}$ to UC.
3. UC sends $\sum_{j=1}^{\ell}\mathrm{Enc}_{\mathcal{E}}\left(\left(P_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)$ to C.

4. C computes $(x_i, y_i)_{1\leq i\leq b} = (k_i G)_{1\leq i\leq b}$
5. C returns $(r_i = x_i \bmod n, t_i = k_i^{-1}(h(m_i) + s \cdot x_i) \bmod n)_{1\leq i\leq b}$

Game 2: This game is the same as **Game 1**, except that all the $P_{k_i^{(j)},j}$ are replaced by 0.

1. C generates $k_i \xleftarrow{\$} \{1, \ldots, n-1\}$ in basis 2^ω
2. C sends $\left(\text{Enc}_{\mathcal{E}}\left((0)_{1\leq i\leq b}\right)\right)_{1\leq j\leq \ell}$ to UC.
3. UC sends $\sum_{j=1}^{\ell} \text{Enc}_{\mathcal{E}}\left((0)_{1\leq i\leq b}\right)$ to C.
4. C computes $(x_i, y_i)_{1\leq i\leq b} = (k_i G)_{1\leq i\leq b}$
5. C returns $(r_i = x_i \bmod n, t_i = k_i^{-1}(h(m_i) + s \cdot x_i) \bmod n)_{1\leq i\leq b}$

Consider a PPT adversary $\mathcal{A}$ having an advantage $Adv_{\mathcal{A}}^{\mathbf{G}_i}$ to produce a forgery in **Game i**, knowing the input, output, transmissions and internal UC processing, such that $Adv_{\mathcal{A}}^{\mathbf{G}_0} = \varepsilon$ and for all games $\mathbf{G}_i$

$$Adv_{\mathcal{A}}^{\mathbf{G}_i} = Pr\left[\mathcal{A}^{\mathbf{G}_i(sk,\cdot)}(pk) = (m, \sigma); \text{Verif}(pk, m, \sigma) = 1\right].$$

Clearly $Adv_{\mathcal{A}}^{\mathbf{G}_1} = Adv_{\mathcal{A}}^{\mathbf{G}_0} = \varepsilon$ since the view of the adversary is exactly the same in the two games. Next, by lemma A.2, we have that $Adv_{\mathcal{A}}^{\mathbf{G}_1} = Adv_{\mathcal{A}}^{\mathbf{G}_2} + \varepsilon(\lambda)$ where $\varepsilon(\lambda)$ is negligible in the security parameter λ. Finally **Game 2** corresponds to a local ECDSA signature generation, therefore $Adv_{\mathcal{A}}^{\mathbf{G}_2} = Adv_{\mathcal{A}}^{\text{ECDSA}}$ and $Adv_{\mathcal{A}}^{\text{ECDSA}}$ is negligible by the UF-CMA hypothesis of ECDSA. □

Lemma A.2. *If $\mathcal{E}$ is IND-CPA, then $Adv_{\mathcal{A}}^{\mathbf{G}_1} = Adv_{\mathcal{A}}^{\mathbf{G}_2} + \varepsilon(\lambda)$ where $\varepsilon(\lambda)$ is negligible.*

Proof. Assume that $\varepsilon(\lambda)$ is not negligible. We can then construct a distinguisher $\mathcal{B}$ having success probability $\geq 1/2 + \varepsilon(\lambda)/\ell$ against the semantic security of Enc.

The proof follows from a simple hybrid argument. We define $\ell + 1$ different games $\mathbf{G}_{1,j'}$ for $j' = 0, \ldots, \ell$ as:

Game (1, j′): This game is the same as **Game 1**, except that all the $P_{k_i^{(j)},j}$'s are replaced by

$$P'_{k_i^{(j)},j} = \begin{cases} 0 & \text{for } j \leq j' \\ P_{k_i^{(j)},j} & \text{for } j > j'. \end{cases}$$

1. C generates $k_i \xleftarrow{\$} \{1, \ldots, n-1\}$ in basis 2^ω
2. C sends $\left(\text{Enc}_{\mathcal{E}}\left(\left(P'_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)\right)_{1\leq j\leq \ell}$ to UC.
3. UC sends $\sum_{j=1}^{\ell} \text{Enc}_{\mathcal{E}}\left(\left(P'_{k_i^{(j)},j}\right)_{1\leq i\leq b}\right)$ to C.
4. C computes $(x_i, y_i)_{1\leq i\leq b} = (k_i G)_{1\leq i\leq b}$
5. C returns $(r_i = x_i \bmod n, t_i = k_i^{-1}(h(m_i) + s \cdot x_i) \bmod n)_{1\leq i\leq b}$

In particular, we have that **Game (1, 0)** is exactly **Game 1** and that **Game (1, ℓ)** is exactly **Game 2**. Therefore there exists an index j_0 such that $\mathcal{A}$ has advantage at least $\varepsilon(\lambda)/\ell$ to distinguish between **Game (1, $\mathbf{j_0}$)** and **Game (1, $\mathbf{j_0 + 1}$)**.

In the following, we construct a distinguisher $\mathcal{B}$ that will include its challenge between these two games.

- $\mathcal{B}$ generates fresh ECDSA keys $(sk'_{\text{ECDSA}}, pk'_{\text{ECDSA}})$
- $\mathcal{B}$ generates $k_i \xleftarrow{\$} \{1, \dots, n-1\}$ in basis 2^ω
- $\mathcal{B}$ generates all the $P'_{k_i^{(j)},j}$'s as

$$P'_{k_i^{(j)},j} = \begin{cases} 0 & \text{for } j \leq j_0 \\ P_{k_i^{(j)},j} & \text{for } j > j_0. \end{cases}$$

- $\mathcal{B}$ asks for a challenge on messages $P_0 = 0$ and $P_1 = P'_{k_i^{(j)},j_0+1}$, and receives a challenge ciphertext Enc (P_β) for an unknown $\beta \in \{0, 1\}$
- Then $\mathcal{B}$ encrypts the $P'_{k_i^{(j)},j}$, but replaces the encryption of $P_{k_i^{(j)},j_0+1}$ by the challenge. We denote by E the resulting set of ciphertexts. Note that if the challenge is encrypting $P_0 = 0$, then the set E corresponds to the set of ciphertexts of **Game (1, $\mathbf{j_0 + 1}$)**, and otherwise it corresponds to the set of ciphertexts of **Game (1, $\mathbf{j_0}$)**.
- Next $\mathcal{B}$ runs $\mathcal{A}$ with E in the ECDSA protocol, and gets its answer (m, σ)
- if Verif$(pk'_{\text{ECDSA}}, m, \sigma) = 1$
  then $\mathcal{B}$ returns 0

  else $\mathcal{B}$ returns 1.

If $b = 0$, this means that the setting was that of **Game (1, $\mathbf{j_0}$)**, and therefore P_1 was encrypted in the challenge; however it was $P_0 = 0$. This contradicts the IND-CPA security of the $\mathcal{E}$ encryption scheme (because ℓ is logarithmic in λ) and concludes the proof. □

A.1.7.3. Malicious adversaries

The proofs presented above consider semi-honest (also known as honest-but-curious) adversaries. In particular, it supposes that the UC behaves as expected during the protocol execution. A malicious, active, UC might divert from the protocol so that the elliptic curve points used for the elliptic-curve point are not correct. With such an attack, the signatures that are published do not follow the ECDSA protocol, and therefore we cannot say they do not reveal enough information for an attacker to break the system.

For example, if the cloud sends back the k_i's as the x coordinates of the obtained points, the client will reveal the k_i's and the attacker will immediately be able to retrieve the secret key used for the signature.

It is possible to modify the protocol so as to ensure resistance against malicious adversaries at a very high cost, but in practice a simpler business-like solution is possible for a lower cost (in practice, most companies will opt for the latter setting). Namely, one can set up traps by including replication

in the k_i's, *i.e.* instead of sampling b different k_i's randomly and batching them all together, to sample b/κ different k_i's and to repeat each of the k_i's in κ different components. (In practice, a few replicas will be enough.) In order to cheat without getting caught in this setting, the malicious UC will have to be consistent with the unknown geometry and his probability of success will therefore be small (similar to the *covert model* of [Lin13]). In order to make it worthless for UC to be malicious, the businesses can set up a contract with the UC in which the cost of getting caught when cheating is more important than the possible gain.

A.2. Implementation and Performances

We implemented prototypes of our delegation protocol. The code was written in C++ using the open-source HElib library (available at `https://github.com/shaih/HElib`) [HS14; HS15]. Our implementation is available under the GNU GPL license at `https://github.com/tricosset/HElib-MP`.

A.2.1. Parameters Constraints and Extending HElib

The P-256 Curve. The commercially available EC solutions usually implement NIST's P-256 EC. Since our goal is to demonstrate the feasibility of our delegation protocol, we chose to focus on the same curve. However, we would like to emphasize that this does not help us: the P-256 curve is quite bad for our setting! On the contrary, curves over binary fields, Koblitz curves or Edwards curves might yield much faster protocols given their small characteristic and their point addition formulae. The study of the fastest EC to be used in combination with homomorphic encryption is certainly an interesting theoretical open problem orthogonal to this work.

Recall that the P-256 curve is the EC

$$(E_{\text{P-256}})\colon y^2 = x^3 - 3x + b \in \mathbb{F}_{p_{\text{P-256}}}, \text{ where}$$

$$\begin{cases} p_{\text{P-256}} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ b &= 41058363725152142129326129780047268409\\ & \quad 1\\ & \quad 14441015993725554835256314039467401291 \end{cases}$$

Extension of HElib to Large p's. The HElib library only handles plaintext space modulus p^r with p and r in single precision. For our protocol however, since we considered the EC P-256, we have to work with plaintext space modulus $p = p_{\text{P-256}}$; we therefore modified HElib to handle that case.[4]

Since larger p is associated with an increase in noise magnitude in the ciphertext and, as a consequence, with a larger ciphertext space modulus q needed to correctly decrypt the ciphertext after the circuit evaluation, we need to carefully choose the BGV parameters, as outlined in the section below.

Selection of BGV Parameters. HElib automatically selects all the parameters for the BGV scheme from the tuple (m, p, L) where m is the index of the cyclotomic polynomial $\Phi_m(x)$, p the plaintext space modulus and L the multiplicative depth of the circuit to be homomorphically evaluated. For the sake of security we need to choose m such that the degree of $\Phi_m(x)$, which is equal to $\phi(m)$

[4] A similar modification had been performed in `https://github.com/dwu4/fhe-si`, but was based on an earlier version of HElib without many of the recent improvements (special primes, etc.).

m	460051	490463	1048576
$\phi(m)$	460050	490462	524288
b	1	490462	262144

Table A.1. – Degree $\phi(m)$ of the cyclotomic polynomial $\Phi_m(x)$, and number b of factors thereof modulo $p = p_{\text{P-256}}$.

where ϕ is the Euler's totient function, is large enough to ensure the desired level of security. To do so, we used the script provided in [APS15] to estimate the bit security of the Learning with Errors instances. Thus for q of 16384 bits, which is the largest size of q considered is this work, and $n > 460000$ we have a security level greater than 130 bits.

As described in Sec. A, our protocol needs to perform batching, *i.e.* needs to embed several plaintext elements per ciphertext. Now, for every (m, p), the cyclotomic polynomial $\Phi_m(x)$ factors modulo p into a product of b irreducible factors $\Phi_m(x) = \prod_{j=1}^{b} F_j(x) \pmod p$ for a $b = b(m, p) \in \{1, \ldots, \deg(\Phi_m(x)) = \phi(m)\}$ where ϕ is Euler's totient function. Unfortunately, not every tuple (p, m) yield an efficient batching; for example $p = p_{\text{P-256}}$ and $m = 460051$ do not allow batching at all — cf. Tab. A.1. In particular, full batching will be possible when $b = \phi(m)$, i.e. when all the roots of $\Phi_m(x)$ are in $\mathbb{Z}_p$. Since $\Phi_m(x)$ divides $x^m - 1$, all the roots of $\Phi_m(x)$ will be in $\mathbb{Z}_p$ when $m \mid p - 1$. Now we have that

$$p_{\text{P-256}} - 1 = 2 \cdot 3 \cdot 5^2 \cdot 17 \cdot 257 \cdot 641 \cdot 1531 \cdot 65537 \cdot 490463 \cdot 6700417 \cdot p',$$

for $p' = 83594504224[...]3916927241$ a large prime. We select $m = 490463$ and by Tab. A.1, this yields a scheme that works with polynomials of degree $490463 - 1 = 490462$, and that can batch 490462 elements. This batching size is quite large but as point computation can often be precomputed (e.g. in signatures, or in computation delegations as in [CLV16]) this issue is generally mitigated.

A.2.2. Windowing

In our delegation protocol, one can select different window sizes ω to encrypt the scalars k_i's. A larger window size will increase the memory used but will decrease the communication cost — cf. Tab. A.3. Also with larger windows, the UC will have to perform less homomorphic computations. The limiting factor is therefore the memory used by the client. Note that in the case the client is very limited in memory (e.g. a smartcard), this can be stored in unprotected memory with a MAC, or even outsourced (e.g. to the phone storage).

More precisely, the k_i's are decomposed in basis 2^ω, and give $2\lambda/\omega$ EC points represented by the two coordinates (x, y) in $\mathbb{F}_{p_{\text{P-256}}}$. Note that the decomposition is easy to compute: it simply consists of the $2\lambda/\omega$ successive sequences of ω bits of the k_i's binary decompositions. The precomputation costs are as follow:

- $(2\lambda)^2/\omega \cdot 2^\omega \cdot 2$ bits in (unprotected) memory storage for the client, and
- $2\lambda/\omega \cdot 2 \cdot \zeta_{\mathcal{E}, p_{\text{P-256}}}$ bits in communication between the client and the UC, where $\zeta_{\mathcal{E}, p_{\text{P-256}}}$ denotes the bits needed to encrypt a coordinate in $\mathbb{F}_{p_{\text{P-256}}}$ with the BGV scheme $\mathcal{E}$.

Moreover, the addition of the encrypted coordinates (which we denote $\text{Enc}_{\mathcal{E}}(x)$ and $\text{Enc}_{\mathcal{E}}(y)$) of the elliptic points received by the UC has a multiplicative depth linear in d_{add} and a number of

Window size	Mult. depth	# Elliptic Curve Additions	Communication Costs C → UC	Communication Costs UC → C	Request Encryption (C)	Point Computation (UC)	Response Decryption (C)
8	2	16	23.8 KB	14.1 KB	2.3 ms	51.0 ms	455.9 μs
8	4	$16+8$	35.6 KB	6.4 KB	3.4 ms	118.4 ms	231.4 μs
8	6	$16+8+4$	47.7 KB	3.0 KB	4.4 ms	190.4 ms	102.2 μs
8	8	$16+8+4+2$	59.7 KB	1.4 KB	5.8 ms	265.5 ms	51.01 μs
8	10	$16+8+4+2+1$	71.5 KB	0.6 KB	7.2 ms	342.6 ms	28.7 μs
16	2	8	11.9 KB	7.0 KB	1.2 ms	25.4 ms	242.8 μs
16	4	$8+4$	12.8 KB	3.2 KB	1.7 ms	63.7 ms	112.0 μs
16	6	$8+4+2$	23.9 KB	1.5 KB	2.4 ms	103.2 ms	51.9 μs
16	8	$8+4+2+1$	29.9 KB	0.7 KB	2.9 ms	136.7 ms	30.4 μs
32	2	4	6.0 KB	3.5 KB	0.7 ms	13.6 ms	131.8 μs
32	4	$4+2$	8.9 KB	1.6 KB	1.0 ms	29.9 ms	53.9 μs
32	6	$4+2+1$	11.9 KB	0.7 KB	1.2 ms	51.5 ms	32.2 μs

Table A.2. – Amortized costs for EC point computation for different window sizes and evaluations of different multiplicative depths — the homomorphic evaluation consists of "# Elliptic Curve Additions" additions of points over the curve P-256. All timings are on a Intel® Core™ i5-4210H CPU and use one core.

Parameters	$\lambda = 128$		
	$\omega = 8$	$\omega = 16$	$\omega = 32$
Enc. points sent	32	16	8
Com. cost (bits)	$64 \cdot \zeta_{\mathcal{E},q}$	$32 \cdot \zeta_{\mathcal{E},q}$	$16 \cdot \zeta_{\mathcal{E},q}$
Outsourced Mem. (Client)	512KB	64MB	2TB
Mult. depth	$5d_{add}$	$4d_{add}$	$3d_{add}$

Table A.3. – Impact of Windowing over storage and communication costs. λ is the bit-security level, ω the bit-size of the window, $\zeta_{\mathcal{E},q}$ is the number of bits required to encrypt an element of $\mathbb{F}_q$, and d_{add} the multiplicative depth of the point addition circuit.

multiplications that is closely related to the representation of those points (in our case, we have $d_{add} = 2$, cf. Sec. A.1.5). It is precisely this depth that determines the size of the BGV parameters $params_{\mathcal{E}}$ and hence, the size of data that is sent. Tab. A.3 shows the impacts of different window sizes over communications and memory usage.

As the UC receives $\ell_{2,n}/\omega$ points to add, his addition circuit has depth $d = \log_2(\ell_{2,n}/\omega)$, which implies that our homomorphic encryption scheme $\mathcal{E}$ must be able to handle d times the depth d_{add} of an homomorphic EC point-addition.

Finally, we can reduce UC's computational cost of the point computation at the expense of an additional work factor for the client by allowing UC to abort the computation prematurely and let the client finish the job. This optimization is discussed in the next Section.

A.2.3. Early Aborts

In our delegation protocol, the UC compute homomorphically the coordinates of

$$kG = \left(\sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} 2^{\omega j} \right) G = \sum_{j=1}^{\ell_{2,n}/\omega} P_{k^{(j)},j} \,,$$

given the encryptions of the coordinates of the $P_{k^{(j)},j}$. In order to reduce the total depth of the point additions computation (namely $\log_2(\ell_{2,n}/\omega) \cdot d_{add}$ using a binary tree), the UC can perform the addition up to a specific multiplicative depth and send back the results (the coordinates of the partial sums) to the client. The latter will have to finish the point addition computation over the plaintexts. This in terms allows to lower the multiplicative depth capability of the BGV scheme at the expense of increasing the number of encrypted coordinates sent to the client. The resulting trade-off depends mainly on the ciphertext expansion. Notice that the more computation does the UC, the larger is the multiplicative depth and the ciphertext expansion, but the less ciphertexts are sent. Several early abortions trade-offs are considered in Tab. A.2.

A.2.4. Full Evaluation

Our prototype was deployed on a commercially available cloud computing service — namely a `c4.8xlarge` AWS instance, with turboboost turned off and running on a single thread, while the client was on a mid-range laptop. We benchmarked on different window sizes $\omega = 8, 16, 32$ and different early aborts (corresponding to resp. multiplicative depths $1 \cdot d_{add}$, $2 \cdot d_{add}$, …, $5 \cdot d_{add}$).

We consider communications costs from the client to the UC (encrypted coordinates of the points to be added), and from the UC to the client (encrypted results). Of course we also consider computational costs for the client and UC. All the results are given in Tab. A.2. Note that the communication and computational costs are for a batch of 1024 point computations, so to obtain the costs per point the figures should be divided by 1024.

One of the most interesting compromises is a windowing size of 16 bits and full additions (no early abort) which can be executed in 74 seconds on a single thread and results in 200Mbits of uploaded data and 3Mbits of downloaded data. In order to use this window size the client also needs to access an internal or locally outsourced memory cache of 32MBytes.

Note that all the operations done by the UC are coordinate wise over vectors of 1024 coordinates given the encryption scheme parameters chosen. Therefore we can suppose that the computation can benefit from a linear gain in a multi-threaded environment with a large amount of threads. It would be possible to send the computation to two `c4.8xlarge` servers which can handle each 36 threads and thus get a reply in roughly a second.

For the client the main computational limit is the throughput at which it can produce the encrypted flow to be sent to the UC and decrypt the flow that comes from the UC. Our client could produce an encryption flow of 800Mbits/s and decrypt an incoming flow of 1.1Gbits/s. Such processing speeds would allow using 8 `c4.8xlarge` instances and retrieve thus up to $4 \cdot 1024$ encrypted points per second.

Note that if the UC is in the cloud and the client requires using it at its maximum throughput, having a constant upload bandwidth usage of 800Mbits/s can be quite costly. The cloud would be more adapted for a sporadic usage. If the processing power is required often, installing computing blades locally (through a LAN or PCI connection) would be a much more interesting strategy.

References

[AFF+13] Carlos Aguilar-Melchor, Simon Fau, Caroline Fontaine, Guy Gogniat, and Renaud Sirdey. "Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain". In: 30.2 (2013). http://sirdeyre.free.fr/Papiers_etc/2013_Recent_advances_in_homomorphic_encryption.pdf, pp. 108–117 (cit. on p. 116).

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of Learning with Errors". In: *Journal of Mathematical Cryptology* 9.3 (2015). https://eprint.iacr.org/2015/046.pdf, pp. 169–203 (cit. on p. 126).

[BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. *Private Database Queries Using Somewhat Homomorphic Encryption*. In: *ACNS 13*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, Heidelberg, June 2013, pp. 102–118 (cit. on p. 116).

[BGMW93] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. *Fast Exponentiation with Precomputation (Extended Abstract)*. In: *EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. LNCS. Springer, Heidelberg, May 1993, pp. 200–207 (cit. on p. 115).

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping*. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on pp. 14, 16, 17, 114, 115, 117, 121).

[BI04] Jean-Claude Bajard and Laurent Imbert. "A full RNS implementation of RSA". In: 53.6 (2004). https://hal.archives-ouvertes.fr/lirmm-00090366/document, pp. 769–774 (cit. on p. 114).

[BL08] Daniel J Bernstein and Tanja Lange. "Analysis and optimization of elliptic-curve single-scalar multiplication". In: *Contemporary Mathematics* 461 (2008). http://www.hyperelliptic.org/EFD/precomp.pdf, pp. 1–20 (cit. on p. 121).

[BLN14] Joppe W Bos, Kristin Lauter, and Michael Naehrig. "Private predictive analysis on encrypted medical data". In: *Journal of biomedical informatics* 50 (2014). http://www.cryptosith.org/papers/biopredict-20141125.pdf, pp. 234–243 (cit. on p. 116).

[BPV98] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. *Speeding up Discrete Log and Factoring Based Schemes via Precomputations*. In: *EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. LNCS. Springer, Heidelberg, May 1998, pp. 221–235 (cit. on p. 115).

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. *Batch Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 315–335 (cit. on pp. 14, 117).

[CKK15] Jung Hee Cheon, Miran Kim, and Myungsun Kim. *Search-and-Compute on Encrypted Data*. In: *FC 2015 Workshops*. Ed. by Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff. Vol. 8976. LNCS. Springer, Heidelberg, Jan. 2015, pp. 142–159 (cit. on p. 116).

[CKL15] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. *Homomorphic Computation of Edit Distance*. In: *FC 2015 Workshops*. Ed. by Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff. Vol. 8976. LNCS. Springer, Heidelberg, Jan. 2015, pp. 194–212 (cit. on p. 116).

[CLV16] Céline Chevalier, Fabien Laguillaumie, and Damien Vergnaud. *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. In: *Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*. https://eprint.iacr.org/2016/309.pdf. Cham: Springer International Publishing, 2016, pp. 261–278. ISBN: 978-3-319-45744-4. URL: http://dx.doi.org/10.1007/978-3-319-45744-4_13 (cit. on pp. 116, 126).

[de 95] Peter de Rooij. *Efficient Exponentiation using Procomputation and Vector Addition Chains*. In: *EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 389–399 (cit. on p. 115).

[Fei86] Joan Feigenbaum. *Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?* In: *CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. LNCS. Springer, Heidelberg, Aug. 1986, pp. 477–488 (cit. on p. 116).

[Gen09] Craig Gentry. *Fully homomorphic encryption using ideal lattices*. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on pp. 9, 14, 17, 114, 117).

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Homomorphic Evaluation of the AES Circuit*. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 850–867 (cit. on pp. 115, 117, 118).

[GLN13] Thore Graepel, Kristin Lauter, and Michael Naehrig. *ML Confidential: Machine Learning on Encrypted Data*. In: *ICISC 12*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. LNCS. Springer, Heidelberg, Nov. 2013, pp. 1–21 (cit. on p. 116).

[HL05] Susan Hohenberger and Anna Lysyanskaya. *How to Securely Outsource Cryptographic Computations*. In: *TCC 2005*. Ed. by Joe Kilian. Vol. 3378. LNCS. Springer, Heidelberg, Feb. 2005, pp. 264–282 (cit. on p. 116).

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. http://diamond.boisestate.edu/~liljanab/MATH308/GuideToECC.pdf. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN: 038795273X (cit. on pp. 116, 120).

[HS14] Shai Halevi and Victor Shoup. *Algorithms in HElib*. In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 554–571 (cit. on pp. 115–118, 125).

[HS15] Shai Halevi and Victor Shoup. *Bootstrapping for HElib*. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 641–670 (cit. on pp. 115–118, 125).

[Knu97] Donald Knuth. *The Art of Computer Programming: Semi-numerical Algorithms, volume Vol. 2*. http://library.aceondo.net/ebooks/Computer_Science/algorithm-the_art_of_computer_programming-knuth.pdf. 1997 (cit. on pp. 114, 116).

[Lin13] Yehuda Lindell. *Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries*. In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–17 (cit. on p. 125).

[LL94] Chae Hoon Lim and Pil Joong Lee. *More Flexible Exponentiation with Precomputation*. In: *CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 95–107 (cit. on p. 115).

[LLN15] Kristin E. Lauter, Adriana López-Alt, and Michael Naehrig. *Private Computation on Encrypted Genomic Data*. In: *LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. LNCS. Springer, Heidelberg, Sept. 2015, pp. 3–27 (cit. on p. 116).

[LN14] Tancrède Lepoint and Michael Naehrig. *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*. In: *AFRICACRYPT 14*. Ed. by David Pointcheval and Damien Vergnaud. Vol. 8469. LNCS. Springer, Heidelberg, May 2014, pp. 318–335 (cit. on p. 116).

[MBG+16] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. *NFLlib: NTT-Based Fast Lattice Library*. In: *CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. LNCS. Springer, Heidelberg, Feb. 2016, pp. 341–356 (cit. on p. 116).

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. *Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 311–343 (cit. on p. 115).

[MKI90] Tsutomu Matsumoto, Koki Kato, and Hideki Imai. *Speeding Up Secret Computations with Insecure Auxiliary Devices*. In: *CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. LNCS. Springer, Heidelberg, Aug. 1990, pp. 497–506 (cit. on p. 116).

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`. 2016 (cit. on pp. 73, 92, 100, 114).

[NLV01] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. *Can homomorphic encryption be practical?* In: *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*. `https://eprint.iacr.org/2011/405.pdf`. 2001, pp. 113–124. URL: `http://doi.acm.org/10.1145/2046660.2046682` (cit. on p. 116).

[NS01] Phong Q. Nguyen and Igor Shparlinski. *On the Insecurity of a Server-Aided RSA Protocol*. In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 21–35 (cit. on p. 116).

[RCB16] Joost Renes, Craig Costello, and Lejla Batina. *Complete Addition Formulas for Prime Order Elliptic Curves*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 403–428 (cit. on pp. 120, 121).

[Riv11] Matthieu Rivain. *Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves*. Cryptology ePrint Archive, Report 2011/338. `http://eprint.iacr.org/2011/338`. 2011 (cit. on p. 119).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (1991), pp. 161–174 (cit. on p. 115).

[Sho15] Victor Shoup. *NTL: A Library for doing Number Theory*. Version 9.4.0. `http://www.shoup.net/ntl`. 2015 (cit. on pp. 115, 118).

[SV10] Nigel P. Smart and Frederik Vercauteren. *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 420–443 (cit. on pp. 117, 118).

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations". In: *Designs, Codes and Cryptography* 71.1 (Apr. 2014), pp. 57–81 (cit. on pp. 14, 117, 118).

[Vau03] Serge Vaudenay. *The Security of DSA and ECDSA*. In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Heidelberg, Jan. 2003, pp. 309–323 (cit. on p. 121).

[VCG+06] Marten Van Dijk, Dwaine Clarke, Blaise Gassend, G Edward Suh, and Srinivas Devadas. "Speeding up exponentiation using an untrusted computational resource". In: *Designs, Codes and Cryptography* 39.2 (2006). `http://www.csl.cornell.edu/~suh/papers/dcc06.pdf`, pp. 253–273 (cit. on p. 116).

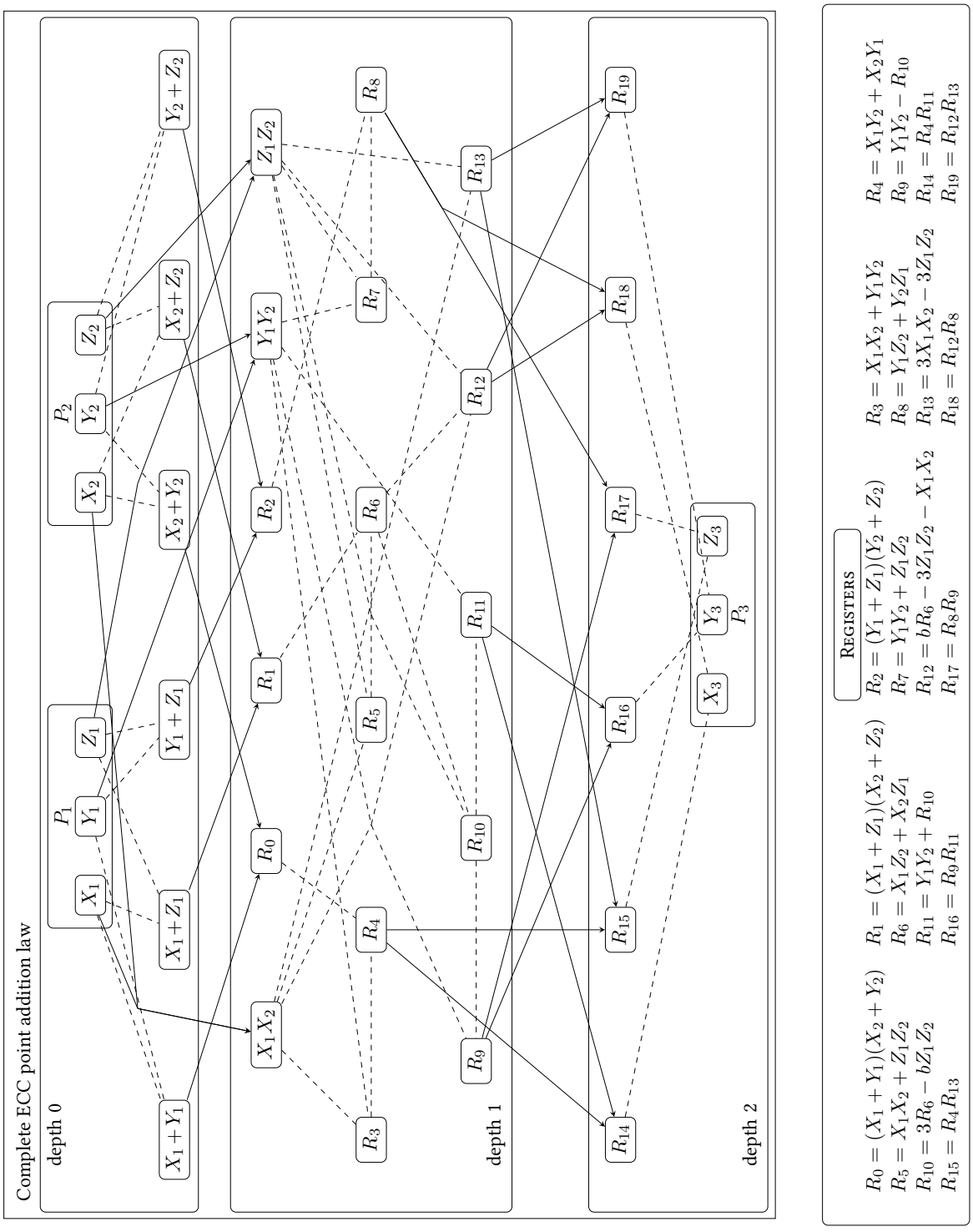
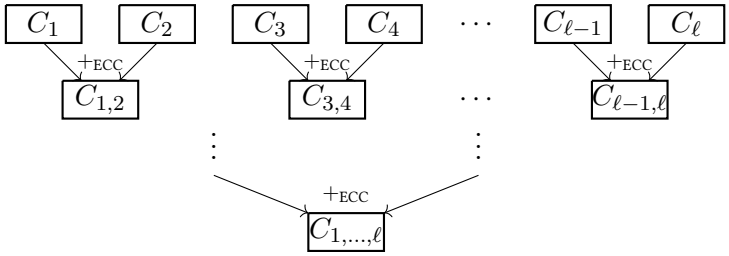


Figure A.2. – Complete addition using [**cryptoeprint:2015:1060**]: only 3 coordinates, multiplicative depth 2, and 12 multiplications (see original algorithm for this). Dashed lines correspond to additions, filled ones to multiplications. We note this algorithm **ECC.Add** and its homomorphic encryption version **HE.ECC.Add**.

ECC point computation outsourcing protocol

Input: The elliptic-curve parameters, a value $\eta \in \mathbb{N}^*$, η scalars k_i, and a FHE scheme $\mathcal{E}$, a batching size b for the FHE scheme, a window size ω, and an EC point representation.
Output: A set of η elliptic-curve points $k_i G$.

1. **Only once:** C computes all the $2^\omega \ell_{2,n}/\omega$ points $\{P_{i,j}\} = \{\left(i \cdot 2^{\omega(j-1)}\right) G\}_{i=0..2^\omega-1}^{j=1..\ell_{2,n}/\omega}$. Noting $\ell = \ell_{2,n}/\omega$, for $k \in \{1, \ldots, n-1\}$ and $(k^{(1)}, \ldots k^{(\ell)})$ the decomposition of k in base 2^ω we have $kG = P_{k^{(1)},1} + \ldots + P_{k^{(\ell)},\ell}$.
2. For each k_i define $(k_i^{(1)}, \ldots, k_i^{(\ell)})$ as its decomposition in basis 2^ω.
3. For each $j \in \{1, \ldots, \ell\}$, C defines $MX_j = (\text{proj}_X(P_{k_1^{(j)},j}), \ldots, \text{proj}_X(P_{k_b^{(j)},j}))$.
4. For each $j \in \{1, \ldots, \ell\}$, C defines $MY_j = (\text{proj}_Y(P_{k_1^{(j)},j}), \ldots, \text{proj}_Y(P_{k_b^{(j)},j}))$.
5. C computes for each $j \in \{1, \ldots, \ell\}$, $CX_j = \text{Enc}_\mathcal{E}(MX_j)$ and $CY_j = \text{Enc}_\mathcal{E}(MY_j)$.
6. C sends $(CX_1, CY_1, \ldots, CX_\ell, CY_\ell)$ to UC.
7. UC generates $CZ_1 = \ldots = CZ_\ell = \text{Enc}_\mathcal{E}((1, \ldots, 1))$, note $C_j = (CX_j, CY_j, CZ_j)$.
8. UC computes homomorphically $C = \textbf{HE.ECC.Add}(C_{j_1}, C_{j_2})$ taking two by two the vectors of encrypted coordinate sets and iterating recursively until there is only one result (see figures A.2).



9. UC sends the encrypted result $C_{1,\ldots,l}$ to C.
10. C decrypts the result and for each $i \in \{1, \ldots b\}$ the coordinates X_i, Y_i, Z_i are used to define a point (x_i, y_i) with $x_i = X_i/Z_i$ and $y_i = Y_i/Z_i$.

Figure A.3. – Full EC point computation delegation protocol with batching

Bibliography

[AAR17] Carlos Aguilar-Melchor, Martin R. Albrecht, and Thomas Ricosset. *Sampling from Arbitrary Centered Discrete Gaussians for Lattice-Based Cryptography*. In: *ACNS 17*. Ed. by Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi. Vol. 10355. LNCS. Springer, Heidelberg, July 2017, pp. 3–19 (cit. on p. 18).

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. *A Subfield Lattice Attack on Overstretched NTRU Assumptions - Cryptanalysis of Some FHE and Graded Encoding Schemes*. In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 153–178 (cit. on p. 69).

[ACLL15] Martin R. Albrecht, Catalin Cocis, Fabien Laguillaumie, and Adeline Langlois. *Implementing Candidate Graded Encoding Schemes from Ideal Lattices*. In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 752–775 (cit. on p. 53).

[ADG+18] Carlos Aguilar-Melchor, Jean-Christophe Deneuville, Philippe Gaborit, Tancrede Lepoint, and Thomas Ricosset. "Delegating Elliptic-Curve Operations with Homomorphic Encryption". In: *4th IEEE Workshop on Security and Privacy in the Cloud* (2018) (cit. on p. 19).

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. *Post-quantum Key Exchange - A New Hope*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 327–343 (cit. on p. 68).

[AE17] Jean-Philippe Aumasson and Guillaume Endignoux. *Improving Stateless Hash-Based Signatures*. Cryptology ePrint Archive, Report 2017/933. `http://eprint.iacr.org/2017/933`. 2017 (cit. on p. 70).

[AFF+13] Carlos Aguilar-Melchor, Simon Fau, Caroline Fontaine, Guy Gogniat, and Renaud Sirdey. "Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain". In: 30.2 (2013). `http://sirdeyre.free.fr/Papiers_etc/2013_Recent_advances_in_homomorphic_encryption.pdf`, pp. 108–117 (cit. on p. 116).

[Ajt96] Miklós Ajtai. *Generating Hard Instances of Lattice Problems (Extended Abstract)*. In: *28th ACM STOC*. ACM Press, May 1996, pp. 99–108 (cit. on pp. 4, 7–9).

[Alb14] Martin R. Albrecht. *dgs — Discrete Gaussians over the Integers*. `https://bitbucket.org/malb/dgs`. 2014 (cit. on pp. 51–53).

[AP13] Jacob Alperin-Sheriff and Chris Peikert. *Practical Bootstrapping in Quasilinear Time*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–20 (cit. on p. 17).

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of Learning with Errors". In: *Journal of Mathematical Cryptology* 9.3 (2015). `https://eprint.iacr.org/2015/046.pdf`, pp. 169–203 (cit. on p. 126).

[AR18] Carlos Aguilar-Melchor and Thomas Ricosset. "CDT-based Gaussian Sampling: From Multi to Double Precision". In: *IEEE Trans. Computers* (2018) (cit. on p. 19).

[Bab85] L Babai. *On Lovasz' Lattice Reduction and the Nearest Lattice Point Problem*. In: *Proceedings on STACS 85 2Nd Annual Symposium on Theoretical Aspects of Computer Science*. New York, NY, USA: Springer-Verlag New York, Inc., 1985 (cit. on p. 64).

[Bab86] L. Babai. "On Lovász' lattice reduction and the nearest lattice point problem". In: *Combinatorica* 6.1 (Mar. 1986), pp. 1–13 (cit. on pp. 13, 14, 64).

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. *Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1006–1018 (cit. on p. 65).

[BCG+14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. *Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers*. In: *SAC 2013*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. LNCS. Springer, Heidelberg, Aug. 2014, pp. 402–417 (cit. on pp. 27, 32).

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. *Random Oracles in a Quantum World*. In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 41–69 (cit. on pp. 13, 63, 64, 70).

[Ber08] Daniel J. Bernstein. "New Stream Cipher Designs". In: ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer-Verlag, 2008. Chap. The Salsa20 Family of Stream Ciphers, pp. 84–97 (cit. on pp. 28, 51).

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. *Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 533–556 (cit. on p. 9).

[BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. *Private Database Queries Using Somewhat Homomorphic Encryption*. In: *ACNS 13*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, Heidelberg, June 2013, pp. 102–118 (cit. on p. 116).

[BGMW93] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. *Fast Exponentiation with Precomputation (Extended Abstract)*. In: *EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. LNCS. Springer, Heidelberg, May 1993, pp. 200–207 (cit. on p. 115).

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping*. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on pp. 14, 16, 17, 114, 115, 117, 121).

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. *SPHINCS: Practical Stateless Hash-Based Signatures*. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 368–397 (cit. on p. 70).

[BHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. *Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme*. In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. 2016, pp. 323–345 (cit. on p. 51).

[BI04] Jean-Claude Bajard and Laurent Imbert. "A full RNS implementation of RSA". In: 53.6 (2004). `https://hal.archives-ouvertes.fr/lirmm-00090366/document`, pp. 769–774 (cit. on p. 114).

[BL08] Daniel J Bernstein and Tanja Lange. "Analysis and optimization of elliptic-curve single-scalar multiplication". In: *Contemporary Mathematics* 461 (2008). `http://www.hyperelliptic.org/EFD/precomp.pdf`, pp. 1–20 (cit. on p. 121).

[BLL+15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. *Improved Security Proofs in Lattice-Based Cryptography: Using the Rényi Divergence Rather Than the Statistical Distance*. In: *ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. Springer, Heidelberg, Nov. 2015, pp. 3–24 (cit. on pp. xix, xxii, 29, 30, 41, 44, 93).

[BLN14] Joppe W Bos, Kristin Lauter, and Michael Naehrig. "Private predictive analysis on encrypted medical data". In: *Journal of biomedical informatics* 50 (2014). `http://www.cryptosith.org/papers/biopredict-20141125.pdf`, pp. 234–243 (cit. on p. 116).

[BPV98] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. *Speeding up Discrete Log and Factoring Based Schemes via Precomputations*. In: *EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. LNCS. Springer, Heidelberg, May 1998, pp. 221–235 (cit. on p. 115).

[Bra12] Zvika Brakerski. *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 868–886 (cit. on pp. 14, 17).

[Bre+06] Richard P Brent et al. *Fast algorithms for high-precision computation of elementary functions*. In: *Proceedings of 7th conference on real numbers and computers (RNC 7)*. 2006, pp. 7–8 (cit. on p. 36).

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. In: *52nd FOCS*. Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 97–106 (cit. on pp. 14, 16).

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. *Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages*. In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 505–524 (cit. on p. 14).

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. *Batch Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 315–335 (cit. on pp. 14, 117).

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. *Short Stickelberger Class Relations and Application to Ideal-SVP*. In: *EUROCRYPT 2017, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, May 2017, pp. 324–348 (cit. on p. 69).

[CFS01] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. *How to Achieve a McEliece-Based Digital Signature Scheme*. In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 157–174 (cit. on p. 70).

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. *An Algorithm for NTRU Problems and Cryptanalysis of the GGH Multilinear Map without a Low Level Encoding of Zero*. Cryptology ePrint Archive, Report 2016/139. `http://eprint.iacr.org/2016/139`. 2016 (cit. on p. 69).

[CKK15] Jung Hee Cheon, Miran Kim, and Myungsun Kim. *Search-and-Compute on Encrypted Data*. In: *FC 2015 Workshops*. Ed. by Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff. Vol. 8976. LNCS. Springer, Heidelberg, Jan. 2015, pp. 142–159 (cit. on p. 116).

[CKL15] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. *Homomorphic Computation of Edit Distance*. In: *FC 2015 Workshops*. Ed. by Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff. Vol. 8976. LNCS. Springer, Heidelberg, Jan. 2015, pp. 194–212 (cit. on p. 116).

[CLV16] Céline Chevalier, Fabien Laguillaumie, and Damien Vergnaud. *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. In: *Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*. `https://eprint.iacr.org/2016/309.pdf`. Cham: Springer International Publishing, 2016, pp. 261–278. ISBN: 978-3-319-45744-4. URL: `http://dx.doi.org/10.1007/978-3-319-45744-4_13` (cit. on pp. 116, 126).

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. *Fully Homomorphic Encryption over the Integers with Shorter Public Keys*. In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 487–504 (cit. on p. 14).

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. *Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 446–464 (cit. on p. 14).

[CRVV15] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. *Efficient software implementation of ring-LWE encryption*. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2015, pp. 339–344 (cit. on p. 49).

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. *Lattice Signatures and Bimodal Gaussians*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 40–56 (cit. on pp. 13, 27, 32, 53).

[de 95] Peter de Rooij. *Efficient Exponentiation using Procomputation and Vector Addition Chains*. In: *EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 389–399 (cit. on p. 115).

[Dev86] L. Devroye. *Non-uniform random variate generation*. Springer-Verlag, 1986 (cit. on pp. 28, 32).

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. "Sampling from discrete Gaussians for lattice-based cryptography on a constrained device". In: *Applicable Algebra in Engineering, Communication and Computing* 25.3 (2014), pp. 159–180 (cit. on pp. 27, 33).

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43 (cit. on p. 14).

[DLL+17] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. *CRYSTALS – Dilithium: Digital Signatures from Module Lattices*. Cryptology ePrint Archive, Report 2017/633. `http://eprint.iacr.org/2017/633`. 2017 (cit. on p. 70).

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. *Efficient Identity-Based Encryption over NTRU Lattices*. In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 22–41 (cit. on pp. xiii, 54, 62, 63, 65, 66, 70).

[DN12a] Léo Ducas and Phong Q. Nguyen. *Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 415–432 (cit. on pp. xix, 32, 33, 36, 37, 40).

[DN12b] Léo Ducas and Phong Q. Nguyen. *Learning a Zonotope and More: Cryptanalysis of NTRUSign Countermeasures*. In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 433–450 (cit. on pp. xiii, xv, 8, 62, 64, 68).

[DP16] Léo Ducas and Thomas Prest. *Fast Fourier Orthogonalization*. In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 191–198. ISBN: 978-1-4503-4380-0. URL: `http://doi.acm.org/10.1145/2930889.2930923` (cit. on pp. xiii, xiv, 62, 67).

[DS05] Jintai Ding and Dieter Schmidt. *Rainbow, a New Multivariable Polynomial Signature Scheme*. In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 164–175 (cit. on p. 70).

[Duc13] Léo Ducas. *Lattice Based Signatures: Attacks, Analysis and Optimization*. Ph.D. Thesis. 2013 (cit. on p. 35).

[EH14] Tim van Erven and Peter Harremoës. "Rényi Divergence and Kullback-Leibler Divergence". In: *IEEE Trans. Information Theory* 60.7 (2014), pp. 3797–3820 (cit. on pp. xix, 30).

[Fei86] Joan Feigenbaum. *Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?* In: *CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. LNCS. Springer, Heidelberg, Aug. 1986, pp. 477–488 (cit. on p. 116).

[FHK+17] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU*. 2017. URL: `https://falcon-sign.info` (cit. on pp. 18, 61).

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. "MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding". In: *ACM Trans. Math. Softw.* 33.2 (June 2007) (cit. on pp. 29, 51, 54).

[FS87] Amos Fiat and Adi Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*. In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194 (cit. on p. 13).

[Gen09] Craig Gentry. *Fully homomorphic encryption using ideal lattices*. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on pp. 9, 14, 17, 114, 117).

[Gen10] Craig Gentry. *Toward Basing Fully Homomorphic Encryption on Worst-Case Hardness*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 116–137 (cit. on p. 14).

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. *Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits*. In: *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. FOCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 40–49 (cit. on p. 9).

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. *Public-Key Cryptosystems from Lattice Reduction Problems*. In: *CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 112–131 (cit. on pp. xiii, 7–9, 13, 62, 64).

[GH11a] Craig Gentry and Shai Halevi. *Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits*. In: *52nd FOCS*. Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 107–109 (cit. on p. 17).

[GH11b] Craig Gentry and Shai Halevi. *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 129–148 (cit. on p. 17).

[GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. *Ring Switching in BGV-Style Homomorphic Encryption*. In: *SCN 12*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. LNCS. Springer, Heidelberg, Sept. 2012, pp. 19–37 (cit. on p. 14).

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Better Bootstrapping in Fully Homomorphic Encryption*. In: *PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. LNCS. Springer, Heidelberg, May 2012, pp. 1–16 (cit. on p. 17).

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Fully Homomorphic Encryption with Polylog Overhead*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 465–482 (cit. on p. 14).

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Homomorphic Evaluation of the AES Circuit*. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 850–867 (cit. on pp. 115, 117, 118).

[GLN13] Thore Graepel, Kristin Lauter, and Michael Naehrig. *ML Confidential: Machine Learning on Encrypted Data*. In: *ICISC 12*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. LNCS. Springer, Heidelberg, Nov. 2013, pp. 1–21 (cit. on p. 116).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. *Trapdoors for hard lattices and new cryptographic constructions*. In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206 (cit. on pp. xii, xiii, 9, 13, 14, 31, 62–65).

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92 (cit. on p. 14).

[Gt15] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 6.0.1. `http://gmplib.org/`. 2015 (cit. on p. 51).

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. *Attribute-based encryption for circuits*. In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 545–554 (cit. on p. 9).

[HHP+03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. *NTRUSIGN: Digital Signatures Using the NTRU Lattice*. In: *CT-RSA 2003*. Ed. by Marc Joye. Vol. 2612. LNCS. Springer, Heidelberg, Apr. 2003, pp. 122–140 (cit. on pp. xiii, 8, 62, 64).

[HL05] Susan Hohenberger and Anna Lysyanskaya. *How to Securely Outsource Cryptographic Computations*. In: *TCC 2005*. Ed. by Joe Kilian. Vol. 3378. LNCS. Springer, Heidelberg, Feb. 2005, pp. 264–282 (cit. on p. 116).

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. `http://diamond.boisestate.edu/~liljanab/MATH308/GuideToECC.pdf`. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN: 038795273X (cit. on pp. 116, 120).

[How07] Nick Howgrave-Graham. *A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU*. In: *CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. LNCS. Springer, Heidelberg, Aug. 2007, pp. 150–169 (cit. on p. 69).

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A Ring-Based Public Key Cryptosystem*. In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. 1998, pp. 267–288 (cit. on pp. 7, 10, 11, 54, 65).

[HS14] Shai Halevi and Victor Shoup. *Algorithms in HElib*. In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 554–571 (cit. on pp. 115–118, 125).

[HS15] Shai Halevi and Victor Shoup. *Bootstrapping for HElib*. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 641–670 (cit. on pp. 115–118, 125).

[Kar16] Charles F. F. Karney. "Sampling Exactly from the Normal Distribution". In: *ACM Trans. Math. Softw.* 42.1 (Jan. 2016), 3:1–3:14 (cit. on pp. 27, 32).

[KF15] Paul Kirchner and Pierre-Alain Fouque. *An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices*. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 43–62 (cit. on p. 68).

[KF17] Paul Kirchner and Pierre-Alain Fouque. *Revisiting Lattice Attacks on Overstretched NTRU Parameters*. In: *EUROCRYPT 2017, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, May 2017, pp. 3–26 (cit. on p. 69).

[Kle00] Philip N. Klein. *Finding the closest lattice vector when it's unusually close*. In: *11th SODA*. Ed. by David B. Shmoys. ACM-SIAM, Jan. 2000, pp. 937–941 (cit. on pp. xiv, 13, 63, 64, 67).

[KLS17] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. *A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model*. Cryptology ePrint Archive, Report 2017/916. `http://eprint.iacr.org/2017/916`. 2017 (cit. on pp. 13, 70).

[Knu97] Donald Knuth. *The Art of Computer Programming: Semi-numerical Algorithms, volume Vol. 2*. `http://library.aceondo.net/ebooks/Computer_Science/algorithm-the_art_of_computer_programming-knuth.pdf`. 1997 (cit. on pp. 114, 116).

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. *Unbalanced Oil and Vinegar Signature Schemes*. In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 206–222 (cit. on p. 70).

[KY76] Donald E. Knuth and Andrew C. Yao. "The Complexity of Nonuniform Random Number Generation". In: *Algorithms and Complexity: New Directions and Recent Results*. Ed. by J. F. Traub. New York: Academic Press, 1976 (cit. on p. 32).

[Lin13] Yehuda Lindell. *Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries*. In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–17 (cit. on p. 125).

[LL94] Chae Hoon Lim and Pil Joong Lee. *More Flexible Exponentiation with Precomputation*. In: *CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 95–107 (cit. on p. 115).

[LLL82] A.K. Lenstra, H.W.jun. Lenstra, and László Lovász. "Factoring polynomials with rational coefficients". In: *Mathematische Annalen* 261.4 (1982), pp. 515–534 (cit. on p. 4).

[LLN15] Kristin E. Lauter, Adriana López-Alt, and Michael Naehrig. *Private Computation on Encrypted Genomic Data*. In: *LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. LNCS. Springer, Heidelberg, Sept. 2015, pp. 3–27 (cit. on p. 116).

[LM08] Vadim Lyubashevsky and Daniele Micciancio. *Asymptotically Efficient Lattice-Based Digital Signatures*. In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Heidelberg, Mar. 2008, pp. 37–54 (cit. on p. 13).

[LN14] Tancrède Lepoint and Michael Naehrig. *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*. In: *AFRICACRYPT 14*. Ed. by David Pointcheval and Damien Vergnaud. Vol. 8469. LNCS. Springer, Heidelberg, May 2014, pp. 318–335 (cit. on p. 116).

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors over Rings*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 1–23 (cit. on pp. 10, 11).

[LW15] Vadim Lyubashevsky and Daniel Wichs. *Simple Lattice Trapdoor Sampling from a Broad Class of Distributions*. In: *PKC 2015*. Ed. by Jonathan Katz. Vol. 9020. LNCS. Springer, Heidelberg, Mar. 2015, pp. 716–730 (cit. on p. 67).

[Lyu08] Vadim Lyubashevsky. *Lattice-Based Identification Schemes Secure Under Active Attacks*. In: *PKC 2008*. Ed. by Ronald Cramer. Vol. 4939. LNCS. Springer, Heidelberg, Mar. 2008, pp. 162–179 (cit. on p. 13).

[Lyu09] Vadim Lyubashevsky. *Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures*. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 598–616 (cit. on p. 13).

[Lyu12] Vadim Lyubashevsky. *Lattice Signatures without Trapdoors*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 738–755 (cit. on p. 13).

[MBG+16] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. *NFLlib: NTT-Based Fast Lattice Library*. In: *CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. LNCS. Springer, Heidelberg, Feb. 2016, pp. 341–356 (cit. on p. 116).

[McE78] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *Deep Space Network Progress Report* 44 (Jan. 1978), pp. 114–116 (cit. on p. 7).

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. *Additively Homomorphic Encryption with d-Operand Multiplications*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 138–154 (cit. on p. 15).

[Mic01] Daniele Micciancio. *Improving Lattice based cryptosystems using the Hermite Normal Form*. In: *Cryptography and Lattices Conference — CaLC 2001*. Vol. 2146. Lecture Notes in Computer Science. Springer-Verlag, 2001, pp. 126–145 (cit. on p. 8).

[Mic02] Daniele Micciancio. *Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions from Worst-Case Complexity Assumptions*. In: *43rd FOCS*. IEEE Computer Society Press, Nov. 2002, pp. 356–365 (cit. on p. 10).

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. *Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 311–343 (cit. on p. 115).

[MKI90] Tsutomu Matsumoto, Koki Kato, and Hideki Imai. *Speeding Up Secret Computations with Insecure Auxiliary Devices*. In: *CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. LNCS. Springer, Heidelberg, Aug. 1990, pp. 497–506 (cit. on p. 116).

[MP12] Daniele Micciancio and Chris Peikert. *Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller*. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 700–718 (cit. on pp. xiv, 67).

[MR04] Daniele Micciancio and Oded Regev. *Worst-Case to Average-Case Reductions Based on Gaussian Measures*. In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 372–381 (cit. on p. 12).

[MR07] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM J. Comput.* 37.1 (Apr. 2007), pp. 267–302 (cit. on pp. 12, 31, 33).

[MT84] G. Marsaglia and W. W. Tsang. "A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions". In: *SIAM J. Sci. Stat. Comput.* 5 (1984), pp. 349–359 (cit. on p. 32).

[MW16] Daniele Micciancio and Michael Walter. *Practical, Predictable Lattice Basis Reduction*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 820–849 (cit. on p. 68).

[MW17] Daniele Micciancio and Michael Walter. *Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time*. In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 455–485 (cit. on pp. xviii, 27, 29, 30, 33, 69, 71).

[Neu51] John von Neumann. "Various Techniques Used in Connection with Random Digits". In: *J. Res. Nat. Bur. Stand.* 12 (1951), pp. 36–38 (cit. on p. 32).

[Ngu99] Phong Q. Nguyen. *Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto'97*. In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 288–304 (cit. on p. 7).

[NIS15] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. `http://dx.doi.org/10.6028/NIST.FIPS.202`. 2015 (cit. on p. 91).

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`. 2016 (cit. on pp. 73, 92, 100, 114).

[NLV01] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. *Can homomorphic encryption be practical?* In: *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*. `https://eprint.iacr.org/2011/405.pdf`. 2001, pp. 113–124. URL: `http://doi.acm.org/10.1145/2046660.2046682` (cit. on p. 116).

[NR06] Phong Q. Nguyen and Oded Regev. *Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 271–288 (cit. on pp. xiii, xv, 7, 8, 13, 62, 64, 68).

[NS01] Phong Q. Nguyen and Igor Shparlinski. *On the Insecurity of a Server-Aided RSA Protocol*. In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 21–35 (cit. on p. 116).

[Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238 (cit. on p. 13).

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. *Enhanced Lattice-Based Signatures on Reconfigurable Hardware*. In: *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. Ed. by Lejla Batina and Matthew Robshaw. Springer Berlin Heidelberg, 2014, pp. 353–370 (cit. on pp. xviii, xxi, xxii, 30, 40, 43, 44).

[Pei10] Chris Peikert. *An Efficient and Parallel Gaussian Sampler for Lattices*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 80–97 (cit. on pp. xiv, xix, xxi, 14, 27, 33, 40, 41, 43, 53, 63, 67).

[Pei16] Chris Peikert. "A Decade of Lattice Cryptography". In: *Foundations and Trends in Theoretical Computer Science* 10.4 (2016), pp. 283–424 (cit. on pp. 3, 10).

[PLP16] Rafaël del Pino, Vadim Lyubashevsky, and David Pointcheval. *The Whole is Less Than the Sum of Its Parts: Constructing More Efficient Lattice-Based AKEs*. In: *SCN 16*. Ed. by Vassilis Zikas and Roberto De Prisco. Vol. 9841. LNCS. Springer, Heidelberg, Aug. 2016, pp. 273–291 (cit. on pp. 13, 63, 64).

[Pre15] Thomas Prest. *Gaussian Sampling in Lattice-Based Cryptography*. Theses. École Normale Supérieure, Dec. 2015. URL: https://tel.archives-ouvertes.fr/tel-01245066 (cit. on p. 66).

[Pre17] Thomas Prest. *Sharper Bounds in Lattice-Based Cryptography Using the Rényi Divergence*. In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Heidelberg, Dec. 2017, pp. 347–374 (cit. on pp. xviii, 30, 40, 68, 69, 92).

[PS08] Xavier Pujol and Damien Stehlé. *Rigorous and Efficient Short Lattice Vectors Enumeration*. In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 390–405 (cit. on p. 35).

[Rab79] M. O. Rabin. *igitalized signatures and public-key functions as intractable as factorization*. Tech. rep. Cambridge, MA, USA, 1979 (cit. on p. 13).

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179 (cit. on pp. 9, 14).

[RCB16] Joost Renes, Craig Costello, and Lejla Batina. *Complete Addition Formulas for Prime Order Elliptic Curves*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 403–428 (cit. on pp. 120, 121).

[Reg05] Oded Regev. *On lattices, learning with errors, random linear codes, and cryptography*. In: *37th ACM STOC*. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93 (cit. on p. 10).

[Riv11] Matthieu Rivain. *Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves*. Cryptology ePrint Archive, Report 2011/338. http://eprint.iacr.org/2011/338. 2011 (cit. on p. 119).

[RRVV14] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. *Compact and Side Channel Secure Discrete Gaussian Sampling*. Cryptology ePrint Archive, Report 2014/591. http://eprint.iacr.org/2014/591. 2014 (cit. on p. 71).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126 (cit. on p. 13).

[Saa16] Markku-Juhani O. Saarinen. *Arithmetic Coding and Blinding Countermeasures for Lattice Signatures: Engineering a Side-Channel Resistant Post-Quantum Signature Scheme with Compact Signatures*. Cryptology ePrint Archive, Report 2016/276. `http://eprint.iacr.org/2016/276`. 2016 (cit. on p. 32).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (1991), pp. 161–174 (cit. on p. 115).

[Sha84] Adi Shamir. "A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem". In: *IEEE Trans. Information Theory* 30.5 (1984), pp. 699–704 (cit. on p. 4).

[Sho15] Victor Shoup. *NTL: A Library for doing Number Theory*. Version 9.4.0. `http://www.shoup.net/ntl`. 2015 (cit. on pp. 115, 118).

[SS11] Damien Stehlé and Ron Steinfeld. *Making NTRU as Secure as Worst-Case Problems over Ideal Lattices*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 27–47 (cit. on pp. xiii, 7, 62, 65, 66).

[SV10] Nigel P. Smart and Frederik Vercauteren. *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 420–443 (cit. on pp. 117, 118).

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations". In: *Designs, Codes and Cryptography* 71.1 (Apr. 2014), pp. 57–81 (cit. on pp. 14, 117, 118).

[Unr17] Dominique Unruh. "Post-Quantum Security of Fiat-Shamir". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 398. URL: `http://eprint.iacr.org/2017/398` (cit. on pp. 13, 70).

[Vau03] Serge Vaudenay. *The Security of DSA and ECDSA*. In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Heidelberg, Jan. 2003, pp. 309–323 (cit. on p. 121).

[VCG+06] Marten Van Dijk, Dwaine Clarke, Blaise Gassend, G Edward Suh, and Srinivas Devadas. "Speeding up exponentiation using an untrusted computational resource". In: *Designs, Codes and Cryptography* 39.2 (2006). `http://www.csl.cornell.edu/~suh/papers/dcc06.pdf`, pp. 253–273 (cit. on p. 116).

[Von51] John Von Neumann. "The general and logical theory of automata". In: *Cerebral mechanisms in behavior* 1.41 (1951), pp. 1–2 (cit. on p. 32).

[Wal74] A. J. Walker. "New fast method for generating discrete random numbers with arbitrary frequency distributions". In: *Electronics Letters* 10 (Apr. 1974), 127–128(1) (cit. on p. 32).

[ZWR+16] Samee Zahur, Xiao Shaun Wang, Mariana Raykova, Adria Gascón, Jack Doerner, David Evans, and Jonathan Katz. *Revisiting Square-Root ORAM: Efficient Random Access in Multi-party Computation*. In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. 2016, pp. 218–234 (cit. on p. 51).

List of Figures

List of Tables

List of Algorithms

Résumé : La cryptographie à base de réseaux euclidiens a généré un vif intérêt durant les deux dernières décennies grâce à des propriétés intéressantes, incluant une conjecture de résistance à l'ordinateur quantique, de fortes garanties de sécurité provenant d'hypothèses de difficulté sur le pire cas et la construction de schémas de chiffrement pleinement homomorphes. Cela dit, bien qu'elle soit cruciale à bon nombre de schémas à base de réseaux euclidiens, la génération de bruit gaussien reste peu étudiée et continue de limiter l'efficacité de cette cryptographie nouvelle.
Cette thèse s'attelle dans un premier temps à améliorer l'efficacité des générateurs de bruit gaussien pour les signatures hache-puis-signe à base de réseaux euclidiens. Nous proposons un nouvel algorithme non-centré, avec un compromis temps-mémoire flexible, aussi rapide que sa variante centrée pour des tables pré-calculées de tailles acceptables en pratique. Nous employons également la divergence de Rényi afin de réduire la précision nécessaire à la double précision standard.
Notre second propos tient à construire Falcon, un nouveau schéma de signature hache-puis-signe, basé sur la méthode théorique de Gentry, Peikert et Vaikuntanathan pour les signatures à base de réseaux euclidiens. Nous instancions cette méthode sur les réseaux NTRU avec un nouvel algorithme de génération de trappes.

Abstract: Lattice-based cryptography has generated considerable interest in the last two decades due to attractive features, including conjectured security against quantum attacks, strong security guarantees from worst-case hardness assumptions and constructions of fully homomorphic encryption schemes. On the other hand, even though it is a crucial part of many lattice-based schemes, Gaussian sampling is still lagging and continues to limit the effectiveness of this new cryptography.
The first goal of this thesis is to improve the efficiency of Gaussian sampling for lattice-based hash-and-sign signature schemes. We propose a non-centered algorithm, with a flexible time-memory tradeoff, as fast as its centered variant for practicable size of precomputed tables. We also use the Rényi divergence to bound the precision requirement to the standard double precision.
Our second objective is to construct Falcon, a new hash-and-sign signature scheme, based on the theoretical framework of Gentry, Peikert and Vaikuntanathan for lattice-based signatures. We instantiate that framework over NTRU lattices with a new trapdoor sampler.