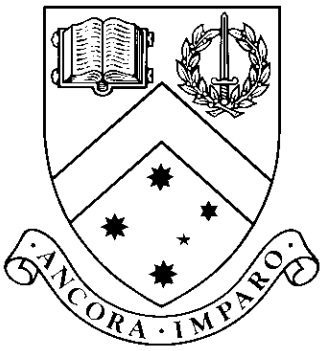
Efficient Implementation Techniques for Lattice-based Cryptosystems

by

Kuo ZHAO (Raymond K. Zhao), MNS, BEng



Thesis

Submitted by Kuo ZHAO (Raymond K. Zhao)

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. Ron Steinfeld

Associate Supervisor: Dr. Amin Sakzad

Faculty of Information Technology
Monash University

February, 2022

Efficient Implementation Techniques for Lattice-based Cryptosystems

Kuo ZHAO (Raymond K. Zhao), MNS, BEng
kuo.zhao@monash.edu
Monash University, 2022

Supervisor: Dr. Ron Steinfeld
Associate Supervisor: Dr. Amin Sakzad

Abstract

Lattice-based cryptosystems are promising candidates for the new standard of post-quantum security due to their efficiency and strong mathematical guarantees. The number of their potential applications has increased in recent years too. Although typical lattice-based cryptosystems are believed to be more efficient than the traditional cryptosystems, their actual performance running on common platforms is believed to be affected heavily by implementation barriers. The security of their implementations (software or hardware) is also an important issue that needs to be considered and addressed properly, since an unprotected implementation is vulnerable to various side-channel attacks including timing and power attacks.

Therefore, this research studies the implementation of various lattice-based cryptosystems and their applications. We aim at finding out better software implementation techniques improving both the efficiency and security of such schemes. A systematic research on software implementation techniques of lattice-based cryptosystems and their applications is carried out during the study. We focus on studying the implementation issues of discrete Gaussian samplers and lattice-based hierarchical identity-based encryption (HIBE) schemes, and make the following three main contributions in this thesis:

For zero-centered discrete Gaussian sampling, we propose a fast, compact, and constant-time implementation of the binary sampling algorithm, originally introduced in the BLISS signature scheme. Our implementation and its analysis adapt the Rényi divergence and the transcendental function polynomial approximation techniques. The efficiency of our scheme is independent of the standard deviation, and we show evidence that our implementations are either faster or more compact than several existing constant-time samplers. In addition, we show the performance of our implementation techniques applied to and integrated with two existing signature schemes: qTesla and Falcon. On the other hand, the convolution theorems are typically adapted

to sample from larger standard deviations, by combining samples with much smaller standard deviations. As an additional contribution, we show better parameters for the convolution theorems.

For arbitrary-centered discrete Gaussian sampling, we propose a compact and scalable rejection sampling algorithm by sampling from a continuous normal distribution and performing rejection sampling on rounded samples. Our scheme does not require precomputations related to any specific discrete Gaussian distributions. Our scheme can sample from both arbitrary centers and arbitrary standard deviations determined on-the-fly at run-time. In addition, we show that our scheme only requires a low number of trials close to 2 per sample on average, and our scheme maintains good performance when scaling up the standard deviation. We also provide a concrete error analysis of our scheme based on the Rényi divergence. We implement our sampler and analyse its performance in terms of storage and speed compared to previous results. Our sampler's run-time is center-independent and is therefore applicable to implementation of convolution-style lattice trapdoor sampling and identity-based encryption resistant against timing side-channel attacks.

For lattice-based hierarchical identity-based encryption schemes, we provide the first complete C implementation and benchmarking of Latte, a promising HIBE scheme endorsed by European Telecommunications Standards Institute (ETSI). We also propose further optimisations for the KeyGen, Delegate, and sampling components of Latte. We adapt the Fast Fourier discrete Gaussian sampling procedures (ffSampling) from the Falcon signature scheme and provide an optimised Fast Fourier $\mathbf{LDL}^*$ decomposition algorithm (ffLDL), one of the key subroutines used by the ffSampling procedure, for lattice basis in Latte HIBE. We show that our optimised ffLDL algorithm is more than 70% faster than a generic naive implementation under 256-bit floating-point arithmetic precision for all Latte parameter sets. In addition, we provide the first provable theoretical error analysis of the ffLDL algorithm and compute the numerical values of the precision bounds for the Latte parameter sets. We evaluate the performance of our optimised Latte implementation. As expected, the KeyGen, Extract, and Delegate components are the most time consuming, with Extract experiencing a 35% decrease in op/s from the first to second hierarchical level at 80-bit security. Our optimised implementation of the Delegate function takes 1 second at this security level on a desktop machine at 4.2GHz, significantly faster than the order of minutes estimated in the ETSI technical report. Furthermore, our optimised Latte Encrypt/Decrypt implementation reaches speeds up to 4.5x faster than the ETSI implementation.

Efficient Implementation Techniques for Lattice-based Cryptosystems

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Kuo ZHAO (Raymond K. Zhao)
February 7, 2022

Publications

Published works (included in the thesis):

- Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. FACCT: FAst, Compact, and Constant-Time Discrete Gaussian Sampler over Integers. (2019). IEEE Transactions on Computers. DOI 10.1109/TC.2019.2940949.
- Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and Scalable Arbitrary-Centered Discrete Gaussian Sampling over Integers. (2020). Proceedings of PQCrypto 2020. DOI 10.1007/978-3-030-44223-1_16.

Preprint (included in the thesis):

- Raymond K. Zhao, Sarah McCarthy, Ron Steinfeld, Amin Sakzad, and Máire O'Neill. Quantum-safe HIBE: does it cost a Latte?. (2021). IACR Cryptology ePrint Archive: Report 2021/222.

Other works during my PhD (not included in the thesis):

- Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. MatRiCT: Efficient, Scalable and Post-Quantum Blockchain Confidential Transactions Protocol. (2019). Proceedings of ACM CCS'19. DOI 10.1145/3319535.3354200.
- Muhammed F. Esgin, Ron Steinfeld, and Raymond K. Zhao. MatRiCT+: More Efficient Post-Quantum Private Blockchain Payments. Accepted by IEEE S&P 2022.
- Muhammed F. Esgin, Ron Steinfeld, Raymond K. Zhao. Efficient Verifiable Partially-Decryptable Commitments from Lattices and Applications. Accepted by PKC 2022.

Acknowledgments

First, I would like to thank my supervisors, Associate Professor Ron Steinfeld and Dr. Amin Sakzad. They provide me lots of guidance throughout the whole PhD journey. Without their insight, patience, and endless support, I would not be able to come this far in my PhD study. Their support is not only limited to the research. They provided me the much-needed emotional help when I was isolated at my home during the COVID lockdowns.

I would like to thank the Faculty of Information Technology and the Graduate Research Office at Monash University for providing me the Monash International Tuition Scholarship to cover the tuition fee during my PhD study. In addition, I would like to express my gratitude to them for offering me the Monash University Graduate Research Completion Award, the Faculty Graduate Research Completion Award, and Faculty of Information Technology International Postgraduate Research Scholarship during the extremely hard time of my thesis extension period in the middle of the COVID lockdown. Furthermore, this research was supported by an Australian Government Research Training Program (RTP) Scholarship.

I would like to thank the mental health nurses working in the Monash University Health Services for helping me coping with the mental health issues caused by the COVID lockdown. I would not even be able to restart working on my PhD thesis without their professional mental health support.

I would like to thank my colleague, Muhammed Esgin, for the collaborations on various other projects during my PhD candidature.

Finally, I would like to thank my parents. Without their understanding and support, I might not even have decided to begin my PhD study in the first place.

Kuo ZHAO (Raymond K. Zhao)

Monash University
February 2022

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The majority of popular traditional public key cryptosystems, such as the de facto industrial standards RSA algorithm [RSA78] and Elliptic-curve cryptography [Kob87, Mil85], was shown to have a compromised security under quantum computing scenarios, due to the evidence of Shor's theoretical work [Sho94] and its recent primitive proof-of-concept [MLL+13] showing that the integer factorisation problem can be solved in polynomial time on quantum computers. On this occasion, the National Institute of Standards and Technology (NIST) in the USA is running a competition (NIST PQC) [NIS16a] evaluating submissions for post-quantum public key cryptosystems. The winning algorithms among all the candidates will hopefully become the new post-quantum security standard. Lattice-based cryptosystems are one of the promising candidates due to the strong mathematical security guarantees and the fact that these cryptosystems have resisted against any known quantum attacks so far. Recently, NIST announced the 3rd round candidates[1] for further evaluations, and five among all the seven 3rd round finalists (71%) are lattice-based cryptosystems. In addition to message encryptions, the lattice-based cryptosystems have various applications in post-quantum security, such as digital signatures [DDLL13], key exchange protocols [ADPS15], and (hierarchical) identity-based encryption (IBE) schemes [ETS19].

A significant advantage of lattice-based cryptosystems is that these cryptosystems typically have a lower time complexity compared to traditional alternatives while still being easy to implement, since the core operations of typical lattice-based cryptosystems belong to either the matrix arithmetic or the polynomial ring arithmetic operations. However, in spite of the theoretical efficiency analysis results, the actual performance of lattice-based cryptosystems running on common platforms are heavily influenced by

[1] https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

their implementation techniques. Alkim et al. have shown the fact that the naive implementation for a lattice-based key exchange protocol only has a "disappointing" performance in the run-time compared to a more optimised implementation [ADPS15]. Meanwhile, an unprotected implementation of lattice-based cryptosystem is also vulnerable to side-channel attacks. For example, various side-channels were exploited in attacking the lattice-based BLISS signature scheme [DDLL13], including the cache [BHLY16], or timing and power [EFGT17]. Therefore, it is essential to study both efficient and secure implementation techniques for lattice-based cryptosystems and their applications.

The general research aim of this study is to propose optimised and secure software implementation techniques for critical steps in various lattice-based cryptosystems and their applications. Our study particularly focuses on the following areas:

Integer Discrete Gaussian Sampler: The integer discrete Gaussian sampler is one of the *key* subroutines in lattice-based schemes based on the (Ring) Learning with Errors problem [Reg05, LPR10] e.g. qTesla digital signature [BAA+17], in lattice trapdoor sampling algorithms [GPV08] e.g. used in Falcon digital signature [PFH+17], and in fully homomorphic encryption schemes e.g. [FV12]. The performance of the integer discrete Gaussian sampler is critical for the run-time speed of such lattice-based cryptosystems [POG15]. In addition, insecure implementations of integer discrete Gaussian samplers have recently become targets of various side-channel attacks [BHLY16, EFGT17].

In this thesis, we study both the *zero-centered* and the *arbitrary-centered* integer discrete Gaussian samplers. This is because lattice-based cryptosystems such as the qTesla [BAA+17] only require sampling from a discrete Gaussian distribution with a *fixed* center (typically, the center is 0), while in lattice trapdoor sampling algorithms [GPV08] the integer discrete Gaussian sampler needs to sample with *arbitrary* centers generated during the run-time.

Lattice-based Hierarchical Identity-based Encryption: To the best of our knowledge, although there exists several practical lattice-based IBE implementation results [DLP14, MSO17, BFRS18, BEP+21], however, prior to our work, the only existing practical lattice-based *hierarchical* IBE (HIBE) implementation result was the *partial* evaluation of the Bonsai-tree [CHKP10] based HIBE scheme called Latte [ETS19], which only gave the performance results of the Encryption and the Decryption algorithms. There was little existing research on the practicality of lattice-based HIBE schemes and therefore it was unclear how to efficiently implement a lattice-based HIBE scheme, especially the Key Delegation algorithm (since the Key Delegation algorithm only exists in HIBE but not in IBE).

Therefore, this research answers the following research questions (RQ):

RQ 1 How to design/develop *efficient* and *constant-time* implementation techniques for the discrete Gaussian sampler with a *fixed* center?

RQ 2 How to design/develop *efficient* and *constant-time* implementation techniques for the discrete Gaussian sampler with *arbitrary* centers generated during the run-time?

RQ 3 How to design/develop *efficient* implementation techniques for *Bonsai-tree* based HIBE schemes?

1.1 Contributions

- For RQ 1, we developed the FACCT sampler (see Chapter 4) to sample from *zero-centered* discrete Gaussian distributions. This constant-time sampler is based on the binary sampling algorithm [DDLL13], but we replaced the previous constant-time Bernoulli sampler [PBY17, EFGT17, BAA+17] by polynomial approximations of $\exp(x)$ to directly evaluate the rejection condition in the binary sampling algorithm instead. To reduce the precision requirements for both the base sampler and the polynomial approximation, we adapted the Rényi divergence [BLL+15, Pre17]. The benchmark results showed that our sampler is either faster or more compact than previous constant-time discrete Gaussian sampler implementations [BAA+17] or countermeasures [PBY17, EFGT17], especially for larger standard deviations. In addition, our sampler has the advantage that the efficiency is independent of the standard deviation, and therefore it is more flexible in implementing schemes requiring samples with different standard deviations. We also adapted our developed techniques to improve the implementation of NIST PQC 2nd round candidates. We showed faster key generation speed in the qTesla signature [ABB+19] after adapting our techniques, and we implemented the $\exp(x)$ subroutine of the Falcon signature[2] [PFH+17] in constant time with very slight overhead.

- For RQ 2, we developed the COSAC sampler (see Chapter 5) to sample from *arbitrary-centered* discrete Gaussian distributions, which is a key subroutine in lattice trapdoor sampling algorithms [GPV08, Pei10]. We generalised the rejection sampler from [Dev86]. Our scheme performs rejection sampling on rounded continuous Gaussian samples [HLS18, ZCHW17] to generate samples from the target discrete Gaussian distribution. Compared with previous arbitrary-centered discrete Gaussian sampling techniques requiring pre-computations

[2] The original implementation of the Falcon signature in NIST PQC 1st round was non-constant time.

[MR18, MW17], our sampling algorithm does not need any pre-computations related to a specific discrete Gaussian distribution or a specific standard deviation, and both the center and the standard deviation can be arbitrary determined on-the-fly at run-time. Our sampler also has a lower number of trials per sample on average (close to 2 per sample) compared to typically about 8–10 per sample on average in the naive rejection sampling algorithm [von51]. The rejection rate of our scheme also decreases when scaling up the standard deviation. In addition, we provided a center-independent run-time implementation (i.e. the run-time is independent of the center), which potentially can be adapted to implement some lattice trapdoor samplers [MP12, Pei10] and applications such as the IBE [BFRS18] in constant time.

- For RQ 3, we optimised and developed the first full practical implementation of Latte [ETS19] (see Chapter 6), a lattice-based HIBE scheme endorsed by the European Telecommunications Standards Institute (ETSI). We adapted techniques from Falcon [PFH+17] and ModFalcon [CPS+20] in order to optimise the implementation, especially the Key Generation algorithm, the Key Delegation algorithm, and the lattice discrete Gaussian sampling subroutine. We also adapted both our FACCT sampler (RQ 1) and COSAC sampler (RQ 2) in order to accelerate the integer discrete Gaussian sampling subroutines in our optimised Latte scheme. In addition, for the (Mod)NTRU basis in Latte, we optimised the Fast Fourier Transform (FFT) based $\mathbf{LDL}^*$ decomposition algorithm (ffLDL) [DP16, PFH+17], which is used in the lattice discrete Gaussian sampling procedure adapted from the Falcon digital signature [PFH+17]. Our optimised ffLDL algorithm achieves a 71.1%–73.4% speedup on average compared to a naive generic ffLDL implementation used by our old Latte implementation for such basis. Furthermore, since the Falcon [PFH+17] only reported the heuristic error bounds of the ffLDL output based on the experimental results with very little technical discussion, we provide the first provable theoretical error analysis results of the ffLDL algorithm, with only a few mild and explicitly stated heuristics, and compute the numerical values of our precision bounds based on the Latte parameter sets. Combining all techniques discussed above, our optimised implementation of the Key Delegation function took 1 second at 80-bit security on a desktop machine at 4.2GHz, significantly faster than the order of minutes estimated in the ETSI technical report [ETS19]. Furthermore, our optimised Latte Encryption and Decryption implementations reach speeds up to 4.5x faster than the ETSI implementation [ETS19].

1.2 Thesis Structure

We show the necessary notations and mathematical backgrounds of this thesis in Chapter 2. In Chapter 3, we review the existing integer discrete Gaussian sampling techniques and practical implementations of lattice-based (hierarchical) identity-based encryption schemes. Our FACCT techniques for zero-centered discrete Gaussian sampling is demonstrated in Chapter 4. Our COSAC techniques for arbitrary-centered discrete Gaussian sampling is demonstrated in Chapter 5. In Chapter 6, we discuss the optimisation techniques and error analysis of the Latte HIBE scheme. We conclude this thesis and discuss both the limitations of our study and potential future works in Chapter 7.

Chapter 2

Preliminaries

2.1 Notations

Arithmetic We denote $\mathbb{Z}^+$ as the positive integer set $\{1, \ldots, \infty\}$, $\mathbb{Z}^-$ as the negative integer set $\{-\infty, \ldots, -1\}$, and $\mathbb{R}^+$ as the set of positive real number $\{x : x \in \mathbb{R}, x > 0\}$, respectively. We define $\lfloor x \rceil$ as the nearest integer to $x \in \mathbb{R}$. We denote $|x|$ as the absolute value of x. The ring of integers mod q is denoted as $\mathbb{Z}_q$. The operator $\oplus$ means XOR. We denote the conjugate of $x \in \mathbb{C}$ as x^*. Let $\gcd(a, b)$ be the greatest common divisor of a, b.

Vectors/Polynomials/Matrices Vectors or, interchangeably through the canonical embedding, polynomials will be denoted by bold small letters like $\mathbf{f}$, matrices $\mathbf{M}$. The transposes of vector $\mathbf{f}$ and matrix $\mathbf{M}$ are denoted as $\mathbf{f}^\mathsf{T}$ and $\mathbf{M}^\mathsf{T}$, respectively. Polynomial ring $\mathbb{Z}[x]/\langle x^N + 1\rangle$ is denoted as $\mathfrak{R}$, and $\mathbb{Z}_q[x]/\langle x^N + 1\rangle$ is denoted as $\mathfrak{R}_q$. The Euclidean norm of a vector/polynomial $\mathbf{f}$ is denoted $\|\mathbf{f}\|$. The Hermitian transpose $\mathbf{f}^*$ of polynomial $\mathbf{f} = f_0 + f_1 x + \cdots + f_{N-1}x^{N-1}$ is defined as $\mathbf{f}^* = f_0 - f_{N-1}x - \cdots - f_1 x^{N-1}$. We denote $\mathbf{M}^*$ as the Hermitian transpose of matrix $\mathbf{M}$ where $(\mathbf{M}^*)_{i,j} = (\mathbf{M}_{j,i})^*$. Let $\det(\mathbf{M})$ be the determinant of matrix $\mathbf{M}$, and let $\mathrm{adj}(\mathbf{M})$ be the adjugate matrix of $\mathbf{M}$. Let $\mathbf{I}_n$ be the $n \times n$ identity matrix, and $\mathbf{0}_n$ be the $n \times n$ zero matrix. The Hermitian product of vectors $\mathbf{a}, \mathbf{b}$ is denoted as $\langle \mathbf{a}, \mathbf{b} \rangle$. The concatenation of several vectors $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N$ will be written as $(\mathbf{f}_1 | \mathbf{f}_2 | \cdots | \mathbf{f}_N)$. A Gram-Schmidt orthogonalised basis is denoted as $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_N\}$. The rounding $\lfloor \mathbf{f} \rceil$ of a polynomial $\mathbf{f}$ is taken to be coefficient-wise rounding. The Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT) of polynomial $\mathbf{f}$ are the evaluations $\mathbf{f}(\omega^i)$ for $i \in \{0, \ldots, N-1\}$, where ω is the $2N$-th complex root of unity in the FFT, and ω is the $2N$-th root of unity mod q in the NTT. The point-wise multiplication of vectors $\mathbf{a}, \mathbf{b}$ is denoted as $\mathbf{a} \odot \mathbf{b}$. The maximal value among n coordinates of scalar $\mathbf{a}$ is denoted as $\max_{i=0}^{n-1} \mathbf{a}_i$, and the minimal value is denoted as $\min_{i=0}^{n-1} \mathbf{a}_i$. The notation $\mathscr{A}(\mathbf{f})$

refers to the anti-circulant matrix associated with polynomial $\mathbf{f}$:

$$\mathcal{A}(\mathbf{f}) = \begin{pmatrix} \mathbf{f}_0 & \mathbf{f}_1 & \dots & \mathbf{f}_{N-1} \\ -\mathbf{f}_{N-1} & \mathbf{f}_0 & \dots & \mathbf{f}_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ -\mathbf{f}_1 & -\mathbf{f}_2 & \dots & \mathbf{f}_0 \end{pmatrix}.$$

Distributions Let $\rho_{\mathbf{c},\sigma}(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right)$ be the n-dimensional (continuous) Gaussian function on $\mathbb{R}^n$ with center $\mathbf{c} \in \mathbb{R}^n$ and standard deviation σ. We denote the discrete Gaussian distribution on lattice Λ with center $\mathbf{c} \in \mathbb{R}^n$ and standard deviation σ by $\mathcal{D}_{\Lambda,\mathbf{c},\sigma}(\mathbf{x}) = \rho_{\mathbf{c},\sigma}(\mathbf{x})/S$, where $S = \rho_{\mathbf{c},\sigma}(\Lambda) = \sum_{\mathbf{k}\in\Lambda} \rho_{\mathbf{c},\sigma}(\mathbf{k})$ is the normalisation factor. We use single value instead of scalar i.e. $\rho_{c,\sigma}(x)$ and $\mathcal{D}_{\Lambda,c,\sigma}(x)$ when the lattice is 1-dimensional. We denote the (1-dimensional) continuous Gaussian (normal) distribution with center c and standard deviation σ by $\mathcal{N}(c,\sigma^2)$, which has the probability density function $\rho_{c,\sigma}(x)/(\sigma\sqrt{2\pi})$. We omit the lattice notation (i.e. $\mathcal{D}_{c,\sigma}$) for $\mathcal{D}_{\mathbb{Z},c,\sigma}$, and we omit the center in notations (i.e. $\rho_\sigma(x)$, $\mathcal{D}_\sigma(x)$, and $\mathcal{D}_{\Lambda,\sigma}(\mathbf{x})$) if the center is zero. We denote $\mathcal{D}_\sigma^+$ as the distribution of $x \hookleftarrow \mathcal{D}_\sigma$ for all $x \in \mathbb{Z}^+ \cup \{0\}$ (i.e. $\mathcal{D}_\sigma^+(x) = \rho_\sigma(x)/\sum_{k\in\mathbb{Z}^+\cup\{0\}} \rho_\sigma(k)$). In addition, we denote the uniform distribution on set S as $\mathcal{U}(S)$ and the Bernoulli distribution with bias p as $\mathcal{B}_p$ (i.e. the probability distribution with $\Pr(X=1)=p$ and $\Pr(X=0)=1-p$). A distribution is B-bounded for some $B \in \mathbb{R}^+$, if its support is in the interval $[-B,B]$ [BLL+15]. Sampling from a distribution $\mathcal{P}$ is denoted by $x \hookleftarrow \mathcal{P}$.

2.2 Mathematical Background

2.2.1 Lattice

An n-dimension lattice $\Lambda(\mathbf{B})$ is the set of all integer linear combinations of some basis set $\mathbf{B}$, where $\mathbf{B} = \{\mathbf{b}_i\}_{i=0}^{n-1} \subseteq \mathbb{R}^n$ and $\mathbf{b}_0, \dots, \mathbf{b}_{n-1}$ are linearly independent: $\Lambda(\mathbf{B}) = \left\{\sum_{i=0}^{n-1} c_i\mathbf{b}_i : c_i \in \mathbb{Z}\right\}$. Lattice Λ is a q-ary lattice if $q\mathbb{Z} \subseteq \Lambda$. For a lattice Λ and any $\epsilon \in \mathbb{R}^+$, we denote the smoothing parameter $\eta_\epsilon(\Lambda)$ as the smallest $s \in \mathbb{R}^+$ such that $\rho_{1/(s\sqrt{2\pi})}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \epsilon$, where Λ^* is the dual lattice of Λ: $\Lambda^* = \{\mathbf{w} \in \mathbb{R}^n : \forall \mathbf{x} \in \Lambda, \mathbf{x}\cdot\mathbf{w} \in \mathbb{Z}\}$ [Pei10]. An upper bound on $\eta_\epsilon(\mathbb{Z})$ is given by [Pei10]: $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2+2/\epsilon)/\pi}$.

Theorem 1 (Adapted from [Pei10], Lemma 2.4)**.** *For any $\epsilon \in (0,1)$ and $c \in \mathbb{R}$, if $\sigma \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$, then $\rho_{c,\sigma}(\mathbb{Z}) = \left[\frac{1-\epsilon}{1+\epsilon}, 1\right] \cdot \rho_\sigma(\mathbb{Z})$, and $\rho_\sigma(\mathbb{Z})$ is approximately $\int_{-\infty}^{\infty} \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{2\pi}$.*

Definition 1 (NTRU Lattice [HHP+03, DLP14]). Let q be a positive integer. Let $\mathbf{f}, \mathbf{g} \in \mathfrak{R}$ and $\mathbf{h} = \mathbf{g}/\mathbf{f} \mod q$. The NTRU lattice associated to $\mathbf{h}$ and q is $\Lambda = \{\mathbf{x} \in \mathfrak{R}^2 : \mathbf{x} \cdot (1, \mathbf{h}) = \mathbf{0} \mod q\}$.

The determinant of the basis of an NTRU lattice is q^N. A basis of the NTRU lattice Λ associated with $\mathbf{h}$ and q is $\mathbf{B} = \begin{pmatrix} -\mathcal{A}(\mathbf{h}) & \mathbf{I}_N \\ q\mathbf{I}_N & \mathbf{0}_N \end{pmatrix}$ [HHP+03, DLP14]. In addition, Another basis of Λ is $\mathbf{S} = \begin{pmatrix} \mathcal{A}(\mathbf{g}) & -\mathcal{A}(\mathbf{f}) \\ \mathcal{A}(\mathbf{G}) & -\mathcal{A}(\mathbf{F}) \end{pmatrix}$ such that $\det(\mathbf{S}) = q^N$ for some $\mathbf{F}, \mathbf{G} \in \mathfrak{R}$ i.e. $\det \begin{pmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{pmatrix} = \mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$ [HHP+03, DLP14].

Definition 2 (ModNTRU Lattice [CKKS19, CPS+20]). Let q be a positive integer. Let $\mathbf{h} \in \mathfrak{R}_q^{d-1}$. The dN-dimensional Module NTRU lattice associated to $\mathbf{h}$ and q is $\Lambda = \{\mathbf{x} \in \mathfrak{R}^d : \mathbf{x} \cdot (1, \mathbf{h}_0, \ldots, \mathbf{h}_{d-2}) = \mathbf{0} \mod q\}$.

The ModNTRU lattice is essentially the generalised NTRU lattice for $d > 2$. A basis of the ModNTRU lattice Λ associated with $\mathbf{h}$ and q is [CKKS19, CPS+20]:

$$\mathbf{B} = \begin{pmatrix} -\mathcal{A}(\mathbf{A}_{d-2}) & \mathbf{0}_N & \ldots & \mathbf{0}_N & \mathbf{I}_N \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\mathcal{A}(\mathbf{h}_0) & \mathbf{I}_N & \ldots & \mathbf{0}_N & \mathbf{0}_N \\ q\mathbf{I}_N & \mathbf{0}_N & \ldots & \mathbf{0}_N & \mathbf{0}_N \end{pmatrix}.$$

In addition, another basis of Λ is:

$$\mathbf{S} = \begin{pmatrix} \mathcal{A}(\mathbf{s}_{0,0}) & \mathcal{A}(\mathbf{s}_{0,1}) & \ldots & \mathcal{A}(\mathbf{s}_{0,d-1}) \\ \mathcal{A}(\mathbf{s}_{1,0}) & \mathcal{A}(\mathbf{s}_{1,1}) & \ldots & \mathcal{A}(\mathbf{s}_{1,d-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}(\mathbf{s}_{d-1,0}) & \mathcal{A}(\mathbf{s}_{d-1,1}) & \ldots & \mathcal{A}(\mathbf{s}_{d-1,d-1}) \end{pmatrix},$$

for some $\mathbf{s}_{i,j} \in \mathfrak{R}$, $0 \le i, j \le d-1$, such that $\mathbf{h}_i = \mathbf{C}_{d-1,i+1}/\mathbf{C}_{d-1,0} \mod q$ for $0 \le i \le d-2$, where $\mathbf{C}$ is the cofactor matrix of $\mathbf{S}' = (\mathbf{s}_{i,j})_{0 \le i,j \le d-1} \in \mathfrak{R}^{(d-1)\times(d-1)}$ [CKKS19, CPS+20]. The determinant of the basis of a ModNTRU lattice is also q^N as in the NTRU lattice i.e. $\det(\mathbf{S}) = q^N$ and $\det(\mathbf{S}') = q$.

2.2.2 Arithmetic Errors

The *absolute* error between values a' and a is $|a' - a|$ where a is accurate and a' is the actual value with errors. Similarly, the *relative* error between a' and a is $\left|\frac{a'-a}{a}\right|$ or $\left|\frac{a'}{a} - 1\right|$.

Let the precision of floating-point number be p bits and let $u = 2^{-p}$. Let $\mathrm{RN}(a)$ be value a rounded to precision p by rounding to the nearest. From the IEEE-754 standard, for $a, b \in \mathbb{R}$ and $z \in \{a, a+b, a-b, a \cdot b, a/b, \sqrt{a}\}$, we have $|\mathrm{RN}(z) - z| \le u|z|$ [MBdD+10].

In complex number arithmetic with precision p, for $z \in \mathbb{C}$, we have $|\mathrm{RN}(z) - z| \leq u|z|$ [MBdD+10]. Muller et al. also provided the absolute error bounds $|\mathrm{RN}(z) - z|$ for $z \in \{a + b, a - b, a \cdot b, a/b\}$ when $a, b \in \mathbb{C}$ are accurate [MBdD+10]. However, for the error analysis in Chapter 6, we need the absolute error of complex number arithmetic when both a, b are actual values with errors. Here, we follow similar approaches to [MBdD+10, BJM+20] and provide the arithmetic error bounds when a, b contain errors.

Addition/Subtraction Let $a, b \in \mathbb{C}$ be the ideal (i.e. accurate) complex numbers and a', b' be the actual complex numbers with the absolute errors $|a' - a| \leq \Delta_a$ and $|b' - b| \leq \Delta_b$. For complex number addition and subtraction $a \pm b$, we have the absolute error $\Delta_\pm(a, b, \Delta_a, \Delta_b) = |\mathrm{RN}(z') - z|$ for $z = a \pm b$ and $z' = a' \pm b'$:

$$\begin{aligned}
\Delta_\pm(a, b, \Delta_a, \Delta_b) &= |\mathrm{RN}(a' \pm b') - (a \pm b)| \\
&\leq |\mathrm{RN}(a' \pm b') - (a' \pm b')| + |(a' \pm b') - (a \pm b)| \\
&\leq |\mathrm{RN}(a' \pm b') - (a' \pm b')| + |a' - a| + |b' - b| \\
&\leq u|a' \pm b'| + \Delta_a + \Delta_b \\
&\leq u(|a \pm b| + |a' - a| + |b' - b|) + \Delta_a + \Delta_b \\
&\leq u(|a| + |b|) + (1 + u)(\Delta_a + \Delta_b).
\end{aligned}$$

Multiplication For complex number multiplication $a \cdot b$, we have the absolute error $\Delta_\times(a, b, \Delta_a, \Delta_b) = |\mathrm{RN}(z') - z|$ for $z = ab$ and $z' = a'b'$:

$$\Delta_\times(a, b, \Delta_a, \Delta_b) = |\mathrm{RN}(a'b') - ab| \leq |\mathrm{RN}(a'b') - a'b'| + |a'b' - ab|.$$

According to [MBdD+10], we have:

$$|\mathrm{RN}(a'b') - a'b'| \leq \sqrt{5}u|a'| \cdot |b'| \leq \sqrt{5}u(|a| + \Delta_a)(|b| + \Delta_b).$$

For $|a'b' - ab|$, we have:

$$\begin{aligned}
|a'b' - ab| &\leq |a'b' - ab'| + |ab' - ab| \\
&= |a' - a| \cdot |b'| + |a| \cdot |b' - b| \\
&\leq \Delta_a|b| + \Delta_b|a| + \Delta_a\Delta_b.
\end{aligned}$$

Therefore,

$$\Delta_\times(a, b, \Delta_a, \Delta_b) \leq \sqrt{5}u|a| \cdot |b| + (\sqrt{5}u + 1)(\Delta_a|b| + \Delta_b|a| + \Delta_a\Delta_b).$$

In addition, when $a \in \mathbb{C}$ and $b \in \mathbb{R}$, we have $|\mathrm{RN}(a'b') - a'b'| \leq u|a'| \cdot |b'|$ instead. Therefore, for the absolute error $\Delta_{\times\mathbb{R}}(a, b, \Delta_a, \Delta_b)$ of complex number multiplication $a \cdot b$ where $a \in \mathbb{C}$, $b \in \mathbb{R}$, we have:

$$\Delta_{\times\mathbb{R}}(a, b, \Delta_a, \Delta_b) \leq u|a| \cdot |b| + (u+1)(\Delta_a|b| + \Delta_b|a| + \Delta_a\Delta_b).$$

Division Similarly, for complex number division a/b, assuming no overflow/underflow occurs, we have the absolute error $\Delta_/(a, b, \Delta_a, \Delta_b) = |\mathrm{RN}(z') - z|$ for $z = a/b$ and $z' = a'/b'$:

$$\Delta_/(a, b, \Delta_a, \Delta_b) = \left|\mathrm{RN}\left(\frac{a'}{b'}\right) - \frac{a}{b}\right| \leq \left|\mathrm{RN}\left(\frac{a'}{b'}\right) - \frac{a'}{b'}\right| + \left|\frac{a'}{b'} - \frac{a}{b}\right|.$$

According to [MBdD+10], we have:

$$\left|\mathrm{RN}\left(\frac{a'}{b'}\right) - \frac{a'}{b'}\right| \leq (5\sqrt{2}(1+6u)u)\frac{|a'|}{|b'|} \leq 5\sqrt{2}(1+6u)u\frac{|a| + \Delta_a}{|b| - \Delta_b}.$$

For $\left|\frac{a'}{b'} - \frac{a}{b}\right|$, we have:

$$\begin{aligned}\left|\frac{a'}{b'} - \frac{a}{b}\right| &= \frac{|a'b - ab'|}{|b'| \cdot |b|} \\ &\leq \frac{|a'b - ab| + |ab - ab'|}{|b'| \cdot |b|} \\ &= \frac{|a' - a| \cdot |b| + |a| \cdot |b - b'|}{|b'| \cdot |b|} \\ &\leq \frac{\Delta_a|b| + \Delta_b|a|}{(|b| - \Delta_b)|b|}.\end{aligned}$$

Therefore,

$$\Delta_/(a, b, \Delta_a, \Delta_b) \leq \frac{5\sqrt{2}(1+6u)u(|a| + \Delta_a) + \Delta_a}{|b| - \Delta_b} + \frac{\Delta_b|a|}{(|b| - \Delta_b)|b|}.$$

In addition, when $a \in \mathbb{C}$ and $b \in \mathbb{R}$, we have $\left|\mathrm{RN}\left(\frac{a'}{b'}\right) - \frac{a'}{b'}\right| \leq u\frac{|a'|}{|b'|}$ instead. Therefore, for the absolute error $\Delta_{/\mathbb{R}}(a, b, \Delta_a, \Delta_b)$ of complex number division a/b where $a \in \mathbb{C}$, $b \in \mathbb{R}$, we have:

$$\Delta_{/\mathbb{R}}(a, b, \Delta_a, \Delta_b) \leq \frac{u|a| + (u+1)\Delta_a}{|b| - \Delta_b} + \frac{\Delta_b|a|}{(|b| - \Delta_b)|b|}.$$

2.2.3 Errors of Fast Fourier Transform

Brisebarre et al. provided the following bound for the error of the Cooley-Tukey Fast Fourier Transform (FFT) [CT65] on the ring $\mathbb{C}[x]/\langle x^N - 1\rangle$ [BJM+20]:

Theorem 2 (Adapted from [BJM+20], Thm. 8)**.** *Assume the floating-point precision is p bits. Let $u = 2^{-p}$. Let $\mathbf{z}$ be the accurate N-point Cooley-Tukey FFT result of $\mathbf{a} \in \mathbb{C}[x]/\langle x^N - 1\rangle$, and let $\mathbf{z}'$ be the actual FFT result with errors. Then,*

$$\|\mathbf{z}' - \mathbf{z}\| \leq \|\mathbf{z}\| \cdot [(1+u)^n(1+g)^{n-2} - 1],$$

where $n = \log_2 N$ and

$$g = \frac{\sqrt{2}}{2}u + \sqrt{5}u\left(1 + \frac{\sqrt{2}}{2}u\right).$$

The exponent $n-2$ of $1+g$ in the inequality above comes from that the complex roots of unity used by the first 2 levels of the Cooley-Tukey FFT on the ring $\mathbb{C}[x]/\langle x^N - 1\rangle$ can be exactly represented, and thus multiplication by them does not introduce errors [BJM+20].

However, for lattice-based cryptosystems, we need the FFT on the ring $\mathbb{C}[x]/\langle x^N + 1\rangle$ instead. For the Cooley-Tukey FFT variant on the ring $\mathbb{C}[x]/\langle x^N + 1\rangle$ e.g. FFT used by Falcon digital signature [PFH+17], only the complex root of unity used by the first level of FFT can be exactly represented. In addition, for the error analysis in Chapter 6, we need the absolute error bound $\max_{i=0}^{N-1} |\mathbf{z}'_i - \mathbf{z}_i|$ of FFT coordinates. Since $\max_{i=0}^{N-1} |\mathbf{z}_i| \leq \|\mathbf{z}\| \leq \sqrt{N} \cdot \max_{i=0}^{N-1} |\mathbf{z}_i|$ [BJM+20], we have the following theorem:

Theorem 3. *Assume the floating-point precision is p bits. Let $u = 2^{-p}$. Let $\mathbf{z}$ be the accurate N-point Cooley-Tukey FFT result of $\mathbf{a} \in \mathbb{C}[x]/\langle x^N + 1\rangle$, and let $\mathbf{z}'$ be the actual FFT result with errors. Then, we have the absolute error Δ_{FFT} of FFT coordinates:*

$$\Delta_{\text{FFT}} = |\mathbf{z}'_i - \mathbf{z}_i| \leq \delta_{\text{FFT}} \cdot \max_{j=0}^{N-1} |\mathbf{z}_j|,$$

$0 \leq i \leq N-1$, where $n = \log_2 N$,

$$\delta_{\text{FFT}} = \sqrt{N}[(1+u)^n(1+g)^{n-1} - 1],$$

and

$$g = \frac{\sqrt{2}}{2}u + \sqrt{5}u\left(1 + \frac{\sqrt{2}}{2}u\right).$$

In addition, for the Euclidean norm of the FFT result, we have:

Theorem 4 (Adapted from [BJM+20])**.** *For N-point FFT result $\mathbf{z}$ of scalar $\mathbf{a}$, we have $\|\mathbf{z}\| = \sqrt{N}\|\mathbf{a}\|$. Thus, $|\mathbf{z}_i| \leq \sqrt{N}\|\mathbf{a}\|$ for $0 \leq i \leq N-1$.*

2.2.4 Divergence

Definition 3 (Relative Error)**.** For two distributions $\mathscr{P}$ and $\mathscr{Q}$ such that $\mathrm{Supp}(\mathscr{P}) = \mathrm{Supp}(\mathscr{Q})$, the relative error between $\mathscr{P}$ and $\mathscr{Q}$ is defined as:

$$\Delta(\mathscr{P}\|\mathscr{Q}) = \max_{x\in\mathrm{Supp}(\mathscr{P})} \frac{|\mathscr{P}(x) - \mathscr{Q}(x)|}{\mathscr{Q}(x)}.$$

Definition 4 (Kullback-Leibler Divergence [PDG14])**.** For two discrete distributions $\mathscr{P}$ and $\mathscr{Q}$ such that $\mathrm{Supp}(\mathscr{P}) \subseteq \mathrm{Supp}(\mathscr{Q})$, the Kullback-Leibler divergence (KLD) is defined as:

$$KL(\mathscr{P}\|\mathscr{Q}) = \sum_{x\in\mathrm{Supp}(\mathscr{P})} \mathscr{P}(x) \ln \frac{\mathscr{P}(x)}{\mathscr{Q}(x)}.$$

Definition 5 (Rényi Divergence [BLL+15, Pre17])**.** For two discrete distributions $\mathscr{P}$ and $\mathscr{Q}$ such that $\mathrm{Supp}(\mathscr{P}) \subseteq \mathrm{Supp}(\mathscr{Q})$, the Rényi divergence (RD) of order $\alpha \in (1, +\infty)$ is defined as:

$$R_\alpha(\mathscr{P}\|\mathscr{Q}) = \left(\sum_{x\in\mathrm{Supp}(\mathscr{P})} \frac{\mathscr{P}(x)^\alpha}{\mathscr{Q}(x)^{\alpha-1}}\right)^{\frac{1}{\alpha-1}}.$$

In addition, for $\alpha = +\infty$, we have:

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

Definition 6 (Max-log Distance [MW17])**.** For two discrete distributions $\mathscr{P}$ and $\mathscr{Q}$ such that $\mathrm{Supp}(\mathscr{P}) = \mathrm{Supp}(\mathscr{Q})$, the max-log distance is defined as:

$$ML(\mathscr{P}\|\mathscr{Q}) = \max_{x\in\mathrm{Supp}(\mathscr{P})} |\ln \mathscr{P}(x) - \ln \mathscr{Q}(x)|.$$

For tighter bounds, we use the following theorems in Chapter 4 and 5:

Theorem 5 (Tail-cut Bound, Adapted from [BLL+15], Thm. 2.11)**.** *Let $\mathscr{D}'_\sigma$ be the B-bounded distribution of $\mathscr{D}_\sigma$ by cutting its tail. For M independent samples, we have $R_\infty((\mathscr{D}'_\sigma)^M\|(\mathscr{D}_\sigma)^M) \le \exp(1)$ if $B \ge \sigma \cdot \sqrt{2\ln(2M)}$.*

Theorem 6 (Relative Error Bound, Adapted from [Pre17], Lemma 3 and Eq. 4)**.** *For two distributions $\mathscr{P}$ and $\mathscr{Q}$ such that* $\mathrm{Supp}(\mathscr{P}) = \mathrm{Supp}(\mathscr{Q})$*, we have:*

$$R_\alpha(\mathscr{P}\|\mathscr{Q}) \le \left(1 + \frac{\alpha(\alpha-1)\cdot(\Delta(\mathscr{P}\|\mathscr{Q}))^2}{2(1-\Delta(\mathscr{P}\|\mathscr{Q}))^{\alpha+1}}\right)^{\frac{1}{\alpha-1}}.$$

The right-hand side is asymptotically equivalent to $1 + \alpha \cdot (\Delta(\mathcal{P}\|\mathcal{Q}))^2/2$ *as* $\Delta(\mathcal{P}\|\mathcal{Q}) \to 0$. *In addition, if a signature scheme using* M *independent samples from* $\mathcal{Q}$ *is* $(\lambda+1)$*-bit secure, then the signature scheme sampling from* $\mathcal{P}$ *will be* λ*-bit secure if* $R_{2\lambda}(\mathcal{P}\|\mathcal{Q}) \leq 1+1/(4M)$.

Typically, we have $M = m \cdot q_s$, where m is the dimension of the lattice and q_s is the number of queries.

Theorem 7 (Adapted from [Pre17], Lemma 4)**.** *For two distributions* $\mathcal{P}$ *and* $\mathcal{Q}$ *such that* $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, *we have:*

$$R_\alpha(\mathcal{P}\|\mathcal{Q}) \leq \left(1 + \frac{\alpha(\alpha-1)\cdot(e^{ML(\mathcal{P}\|\mathcal{Q})}-1)^2}{2(2-e^{ML(\mathcal{P}\|\mathcal{Q})})^{\alpha+1}}\right)^{\frac{1}{\alpha-1}}.$$

The right-hand side is asymptotically equivalent to $1+\alpha\cdot(ML(\mathcal{P}\|\mathcal{Q}))^2/2$ *as* $ML(\mathcal{P}\|\mathcal{Q}) \to 0$.

2.2.5 FFT Sampling of Lattice Discrete Gaussian

In order to sample from the lattice Gaussian distribution $\mathcal{D}_{\Lambda(\mathbf{B}),\mathbf{c},\sigma}$, typically a discrete Gaussian sampler requires either the Gram-Schmidt Orthogonalisation (GSO) of the basis $\mathbf{B}$ [Kle00, GPV08] or the largest singular value $s_1(\mathbf{B})$ [Pei10]. For the basis $\mathbf{B} \in \mathfrak{R}^{d\times d}$, Ducas and Prest provided a fast orthogonalisation algorithm utilising the $\mathbf{LDL}^*$ decomposition and Fast Fourier Transform (FFT) [DP16].

Definition 7 (Gram-Schmidt Orthogonal Decomposition [DP16])**.** Let $\mathbf{B} \in \mathfrak{R}^{d\times d}$ be a full-rank matrix. There exists a Gram-Schmidt Orthogonal (GSO) Decomposition $\mathbf{B} = \mathbf{L}\cdot\tilde{\mathbf{B}}$, where $\mathbf{L}$ is unit lower triangular and rows $\tilde{\mathbf{B}}_i$ of $\tilde{\mathbf{B}}$ are pairwise orthogonal.

Definition 8 ($\mathbf{LDL}^*$ Decomposition [DP16])**.** Let the full-rank Gram matrix $\mathbf{G} = \mathbf{BB}^*$ where $\mathbf{B} \in \mathfrak{R}^{d\times d}$. There exists an $\mathbf{LDL}^*$ Decomposition $\mathbf{G} = \mathbf{LDL}^*$, where $\mathbf{L}$ is a lower triangular matrix with 1 on its diagonal and $\mathbf{D}$ is a diagonal matrix.

In addition, if $\mathbf{B}$ has the GSO decomposition $\mathbf{B} = \mathbf{L}\cdot\tilde{\mathbf{B}}$, then $\mathbf{L}\cdot(\tilde{\mathbf{B}}\tilde{\mathbf{B}}^*)\cdot\mathbf{L}^*$ is the $\mathbf{LDL}^*$ decomposition of $\mathbf{G} = \mathbf{BB}^*$ [DP16]. Therefore, the diagonal of $\mathbf{D}$ in the $\mathbf{LDL}^*$ decomposition of $\mathbf{G} = \mathbf{BB}^*$ is essentially the square of the Euclidean norms of GSO vectors $\|\tilde{\mathbf{B}}_i\|^2$.

For the ring $\mathbb{C}[x]/\langle x^N+1\rangle$ where N is power of 2, Ducas and Prest showed that the ffLDL algorithm in Algorithm 2.1 can perform the $\mathbf{LDL}^*$ decomposition on the input $\mathbf{G} = \mathbf{BB}^* \in (\mathbb{C}[x]/\langle x^N+1\rangle)^{d\times d}$ in the FFT domain [DP16]. The leaf values of the tree T are $\|\tilde{\mathbf{B}}_i\|^2$.

Definition 9 (splitfft)**.** The splitfft function in Algorithm 2.1 is essentially the Gentleman-Sande butterfly [GS66]:

$$(\mathbf{d}_0)_j = \frac{1}{2}[(\mathbf{D}_{i,i})_{2j} + (\mathbf{D}_{i,i})_{2j+1}], \quad (\mathbf{d}_1)_j = \frac{1}{2}[(\mathbf{D}_{i,i})_{2j} - (\mathbf{D}_{i,i})_{2j+1}]\omega^{-\text{bitrev}(n/2+j)},$$

for $j \in \{0, \ldots, n/2-1\}$, where ω is the $2N$-th complex root of unity and bitrev is the bit reverse function.

Algorithm 2.1 The ffLDL algorithm from [DP16, PFH+17].

Input: Gram matrix $\mathbf{G} \in (\mathbb{C}[x]/\langle x^n+1\rangle)^{d\times d}$ in the FFT domain.
Output: Tree T.

function ffLDL($\mathbf{G}$)
  if $n = 1$ **then**
    T.value $\leftarrow \mathbf{G}_{0,0}$.
  else
    $\mathbf{L} \leftarrow \mathbf{I}_d, \mathbf{D} \leftarrow \mathbf{0}_d$.
    for $i = 0$ **to** $d-1$ **do**
      for $j = 0$ **to** $i-1$ **do**
        $\mathbf{L}_{i,j} \leftarrow \frac{1}{\mathbf{D}_{j,j}}\left(\mathbf{G}_{i,j} - \sum_{k<j} \mathbf{L}_{i,k} \odot \mathbf{L}^*_{j,k} \odot \mathbf{D}_{k,k}\right)$.
      end for
      $\mathbf{D}_{i,i} \leftarrow \mathbf{G}_{i,i} - \sum_{j<i} \mathbf{L}_{i,j} \odot \mathbf{L}^*_{i,j} \odot \mathbf{D}_{j,j}$.
    end for
    T.value $\leftarrow \mathbf{L}$.
    for $i = 0$ **to** $d-1$ **do**
      $\mathbf{d}_0, \mathbf{d}_1 \leftarrow$ splitfft($\mathbf{D}_{i,i}$).
      $\mathbf{G}' = \begin{pmatrix} \mathbf{d}_0 & \mathbf{d}_1 \\ \mathbf{d}_1^* & \mathbf{d}_0 \end{pmatrix}$.
      T.child$_i \leftarrow$ ffLDL($\mathbf{G}'$).
    end for
  end if
  return T.
end function

The Falcon digital signature [PFH+17] utilised the ffLDL tree T and implemented a fast discrete Gaussian sampler over lattice $\Lambda(\mathbf{B})$. In order to sample from $\mathcal{D}_{\mathbf{c}+\Lambda(\mathbf{B}),\sigma}$ for q-ary lattice $\Lambda(\mathbf{B})$, the FFT sampling procedure [PFH+17] works as follows:

1. Call Tree($\mathbf{B}, \sigma$) function from Algorithm 2.2. The Tree function in Algorithm 2.2 performs the ffLDL decomposition of $\mathbf{G} = \mathbf{B}\mathbf{B}^*$ by calling the ffLDL algorithm in Algorithm 2.1 and then normalises the leaf values by using σ.

2. Let $\mathbf{t} = \mathbf{c} \cdot \mathbf{B}^{-1}$. Call ffSampling(FFT($\mathbf{t}$), T) function from Algorithm 2.3. The ffSampling function is essentially a randomised variant of the Fast Fourier nearest plan algorithm [DP16]. The ffSampling algorithm returns a vector $\mathbf{z}$ in the FFT

domain such that $\mathbf{zB}$ follows the distribution $\mathscr{D}_{\Lambda(\mathbf{B}),\mathbf{tB},\sigma}$ [PFH+17]. Thus, for q-ary lattice $\Lambda(\mathbf{B})$, $(\mathbf{t}-\mathbf{z})\mathbf{B}$ follows the distribution $\mathscr{D}_{\mathbf{c}+\Lambda(\mathbf{B}),\sigma}$ [GPV08].

Algorithm 2.2 The ffSampling Tree computation algorithm [PFH+17].

Input: $\mathbf{B}, \sigma$.
Output: Tree T.

function Tree($\mathbf{B}, \sigma$)
  $\mathbf{G} \leftarrow \mathbf{B}\mathbf{B}^*$.
  $T \leftarrow$ ffLDL(FFT($\mathbf{G}$)).
  For each leaf of T, leaf.value $\leftarrow \sigma/\sqrt{\text{leaf.value}}$.
  return T.
end function

Algorithm 2.3 The ffSampling algorithm [PFH+17].

Input: $\mathbf{t} = (\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{d-1})$ in FFT format, tree T.
Output: $\mathbf{z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{d-1})$ in FFT format.

function ffSampling($\mathbf{t}, T$)
  if $n = 1$ **then**
    $\sigma' \leftarrow T$.value.
    $z_0 \leftarrow \mathscr{D}_{\sigma',t_0}$.
    $z_1 \leftarrow \mathscr{D}_{\sigma',t_1}$.
    return $\mathbf{z} = (z_0, z_1)$.
  else
    $m \leftarrow$ number of children of T.
    for $j = m-1$ **downto** 0 **do**
      $T_j \leftarrow j$-th child of T.
      $\mathbf{t}'_j \leftarrow \mathbf{t}_j + \sum_{i=j+1}^{m}(\mathbf{t}_i - \mathbf{z}_i) \cdot T.\text{value}_{i,j}$.
      $\mathbf{t}'_j \leftarrow \text{splitfft}(\mathbf{t}'_j)$.
      $\mathbf{z}'_j \leftarrow \text{ffSampling}(\mathbf{t}'_j, T_j)$.
      $\mathbf{z}_j \leftarrow \text{mergefft}(\mathbf{z}'_j)$.
    end for
    return $\mathbf{z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1})$.
  end if
end function

2.2.6 (Hierarchical) Identity-based Encryption

The identity-based encryption (IBE) [Sha84] is the public key encryption scheme where the user's public key is (derived from) an arbitrary string i.e. identity. In an IBE system, the key manager will first generate the master private key. The user can register and request their private key from the key manager by using their identity string. The key manager will extract the user's private key by using the master private key and the user's identity string. Users can then perform encryption/decryption by using their key pairs. More specifically, an IBE scheme typically contains the following functions [BF03]:

KeyGen: The master key generator establishes the master public and private keys.

Extract: The extractor uses the master public/private key pair and the user's identity string to extract and share user public/private keys.

Encrypt/Decrypt: Users can use their public key derived from their identity string and their extracted private key to perform encryption and decryption.

Hierarchical identity-based encryption (HIBE) schemes were introduced by [HL02] and can be considered a generalisation of an IBE scheme to multiple levels. There can be multiple (sub) key managers at each level and each (sub) key manager has their own master public/private key pair. A key manager can use their own master private key and the identity string of a sub key manager at next level to delegate a master public/private key pair to that sub key manager. That sub key manager can use their received master key pair to perform the Extract function as in a normal IBE for users at this level, or delegate a master public/private key pair to another sub key manager at next level. Users at each level can register and request their private key by using their identity string at any (sub) key manager on the same level. More specifically, in addition to the 4 functions above in an IBE scheme, an HIBE has an extra **Delegate** function. Through a delegation function, the master key generator creates a public/private key pair for the sub key manager. This gives it the ability to delegate further key pairs, and extract user private keys at that level.

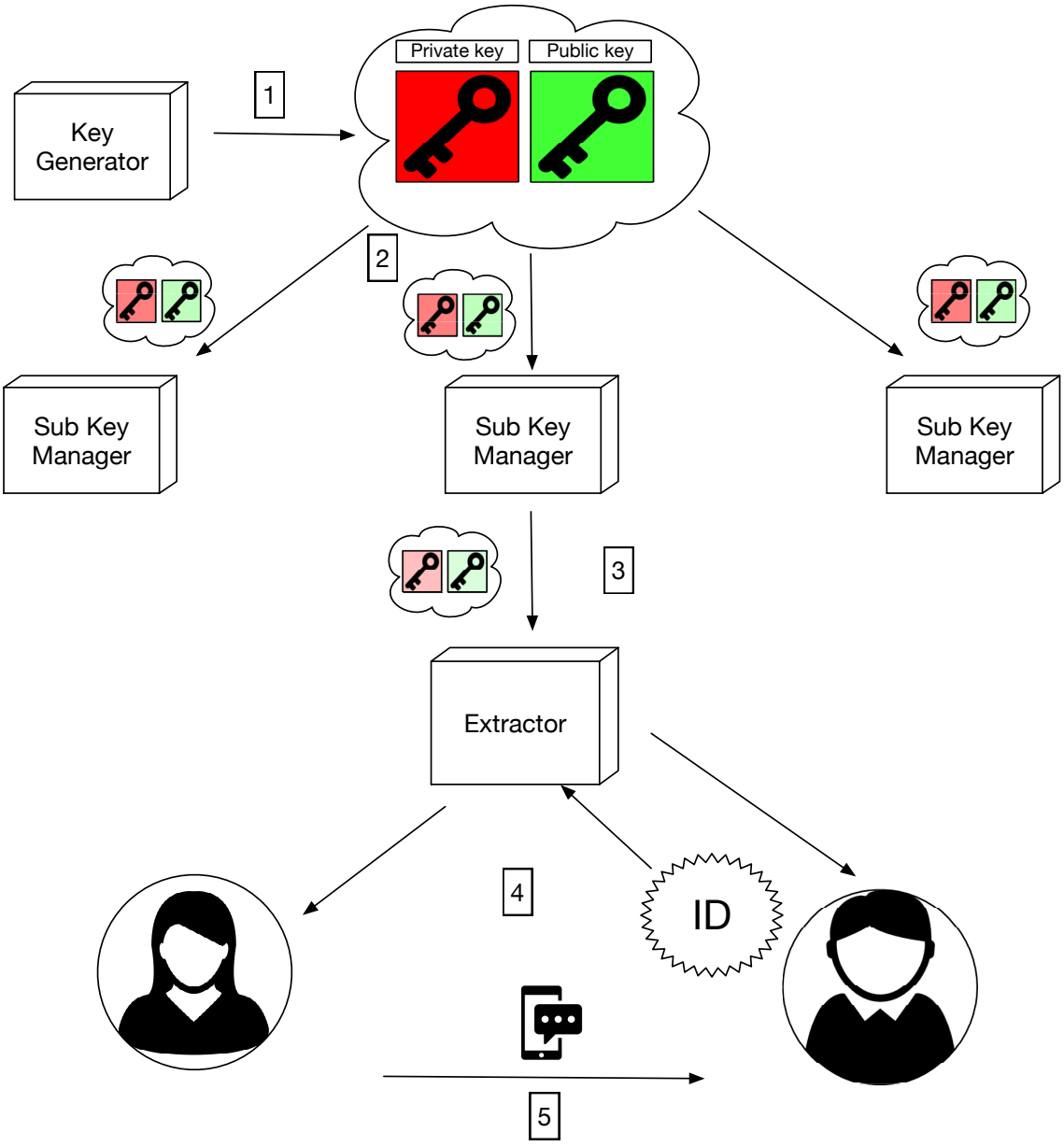
Figure 2.1 shows how each component in a 2-level HIBE scheme interact with each other. Label 1 in Figure 2.1 performs the KeyGen function, Label 2, 3 perform the Delegate function, Label 4 performs the Extract function, and Label 5 perform the Encrypt/Decrypt functions.

Table 2.1 indicates the HIBE functions that can be performed at each hierarchical level in a 2-level HIBE scheme. We consider a user at level ℓ can extract the user's private key by using the master key at level $\ell - 1$. For example, a single-level IBE is the same as considering only level 1 without the delegation. Delegation is from level $\ell - 1$ to level ℓ. A user at level $\ell + 1$ can then extract the user's private key by using this delegated key, and this user's key pair can be used to encrypt and decrypt at level $\ell + 1$.

Table 2.1: Explanation of Notational Practice of 2-level HIBE Functions.

Level ℓ	**Function**
$\ell = 0$	Master KeyGen $\mathbf{S}_0$
$\ell = 1$	Extracting with $\mathbf{S}_0 \rightarrow$ Enc/Dec
	Delegating to $\mathbf{S}_1$
$\ell = 2$	Extracting with $\mathbf{S}_1 \rightarrow$ Enc/Dec

Figure 2.1: A 2-level HIBE scheme [ZMS+21].



2.2.7 Miscellaneous

Cramer's Rule Cramer's rule [CG50] is used for solving systems of linear equations. Consider a system of N equations with N unknowns, represented as $\mathbf{Ax} = \mathbf{b}$. Cramer's rule states that the solution can be written as $\mathbf{x}_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})}$, where $\mathbf{A}_i$ is the matrix formed by replacing the i-th column of $\mathbf{A}$ by the column vector $\mathbf{b}$.

Chapter 3

Literature Review

3.1 Discrete Gaussian Sampler

This section reviews the discrete Gaussian sampling techniques with regards to lattice-based cryptosystems. Since the majority of lattice-based cryptosystems either directly uses samples from the zero-centered discrete Gaussian distribution $\mathscr{D}_\sigma$ e.g. the BLISS digital signature [DDLL13], or employs an arbitrary-centered discrete Gaussian sampler of the distribution $\mathscr{D}_{c,\sigma}$ as the subroutine to sample from a discrete Gaussian distribution on some lattices e.g. [GPV08], we focus on the techniques to sample from a discrete Gaussian distribution over integers in this section.

For the discussion with regards to the qTesla digital signature scheme [BAA+17, ABB+19], since its sampling algorithm is drastically different between the NIST PQC 1st round implementation [BAA+17] and the NIST PQC 2nd round implementation [ABB+19], we use the notations qTesla-R1 [BAA+17] and qTesla-R2 [ABB+19] to distinguish these two works.

3.1.1 Cumulative Distribution Table

The Cumulative Distribution Table (CDT) method (or the Inversion method) [Dev86], was adapted by [Pei10] in lattice-based cryptosystems. Let denote $\Phi_{c,\sigma}$ as the Cumulative Distribution Function (CDF) of $\mathscr{D}_{c,\sigma}$ for some fixed center c: $\Phi_{c,\sigma}(x) = \sum_{k=-\infty}^{x} \mathscr{D}_{c,\sigma}(k)$, where $x, k \in \mathbb{Z}$. In order to generate $x \hookleftarrow \mathscr{D}_{c,\sigma}$, the CDT algorithm performs a table search to find such an x that $\Phi_{c,\sigma}(x-1) < r \leq \Phi_{c,\sigma}(x)$ for some random real $r \in [0, 1]$.

In the zero-centered case i.e. $c = 0$, since $\mathscr{D}_\sigma$ is symmetrical with respect to 0, one can fold $\mathscr{D}_\sigma$ to reduce the table storage by half [DB15]. Let define the folded discrete

Gaussian distribution:

$$\mathscr{D}'_\sigma(x) = \begin{cases} \mathscr{D}_\sigma(x) & x = 0 \\ 2 \cdot \mathscr{D}_\sigma(x) & x > 0 \end{cases},$$

and its CDF $\Phi'_\sigma(x) = \sum_{k=0}^{x} \mathscr{D}'_\sigma(x)$, where $x, k \in \mathbb{Z}^+$. Then, one can sample from $\mathscr{D}_\sigma$ by performing the CDT algorithm on the table Φ'_σ and applying a sign bit with equal probability.

The CDF $\Phi_{c,\sigma}$ in actual implementations typically has a bounded support $x \in [c - \tau\sigma, c + \tau\sigma]$ (or $x \in [0, \tau\sigma]$ for the folded CDF Φ'_σ), where τ is the tail-cut factor (typically, about 10–12). In addition, since $\mathscr{D}_{c,\sigma}$ is a light-tailed distribution, the relative error of the tail in the CDT may increase significantly. To reduce the error, Pöppelmann et al. suggested computing the CDT in a reversed order [PDG14]: for the folded discrete Gaussian distribution $\mathscr{D}'_\sigma$, one may let $\mathrm{CDT}[i] - \mathrm{CDT}[i+1] = \mathscr{D}'_\sigma(i)$ for $i \in [0, \tau\sigma]$ to ensure the CDT only enlarges the relative error by a factor of about σ.

To implement the CDT table search in constant time, a simple but inefficient method is performing a full-table linear search algorithm with $\mathcal{O}(\tau\sigma)$ time complexity by using constant-time comparisons between the randomness and each table entry [BCNS15]. Meanwhile, it is clear that $\Phi_{c,\sigma}$ is a strictly increasing function. Therefore, one may adapt a binary search algorithm to reduce the time complexity to $\mathcal{O}(\log_2 \tau\sigma)$, which is much faster than the $\mathcal{O}(\tau\sigma)$ linear search. However, a naive binary search implementation may take less than $\log_2 \tau\sigma$ steps since it will terminate immediately when the randomness is equal to a table entry in the CDT. Howe et al. suggested a binary search algorithm over the CDT with constant number of iterations $\mathcal{O}(\log_2 \tau\sigma)$ on hardware [HKR+18]. However, the memory access in this approach is not constant, which might cause potential cache timing leakage in software implementations [KRR+18].

For smaller standard deviations, a more recent approach used in the qTesla-R2 digital signature [ABB+19] suggested generating a batch of randomness and performing a constant-time sorting algorithm on the array containing all the randomness accompanied by the CDT table entries. In the sorted array, the CDT table entries at their new positions will separate the array into multiple intervals, and by definition of the CDT, one can get the associated sample values at each interval.

In addition, for the arbitrary-centered discrete Gaussian distribution $\mathscr{D}_{c,\sigma}$, a variant of the CDT method [Dev86] with multiple pre-computed tables was suggested by [MAR17, MR18]. These algorithms will have two phases: online and offline. To be more specific, for $c \in [0, 1)$, during the offline phase, the algorithm pre-computes multiple CDTs of $\mathscr{D}_{i/n,\sigma}$, where $i \in \{0, \ldots, n-1\}$ and $n \in \mathbb{Z}^+$ is sufficiently large. During the online phase, the algorithm picks a sample generated from either $\mathscr{D}_{\lfloor n(c-\lfloor c \rfloor) \rfloor/n,\sigma}$ or $\mathscr{D}_{\lceil n(c-\lfloor c \rfloor) \rceil/n,\sigma}$ as the output. Although the algorithm is fast, however, σ is fixed during the offline computation

and thus this algorithm cannot support sampling from $\mathscr{D}_{c,\sigma}$ with both arbitrary c and σ determined on-the-fly at run-time. Another issue is that the pre-computation table size grows significantly when scaling up σ and therefore the algorithm is not scalable.

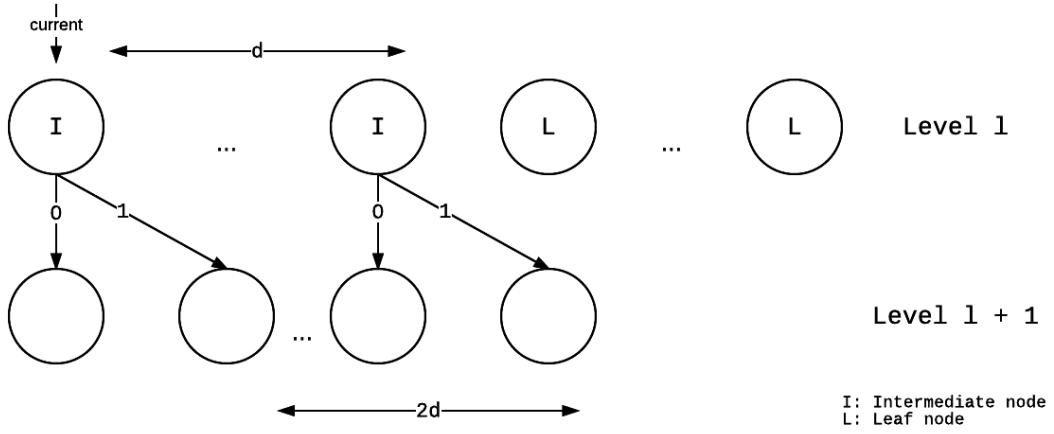
3.1.2 Knuth-Yao Algorithm

The Knuth-Yao sampling algorithm [KY76] generates new samples by traversing a Discrete Distribution Generating (DDG) tree, which is constructed directly from the images of $\mathscr{D}_{c,\sigma}$:

Definition 10 (DDG Tree [KY76])**.** A DDG tree of $\mathscr{D}_{c,\sigma}$ is a binary tree where:

- Each non-leaf node has exactly two children.
- Each leaf node has a value $x \in \mathbb{Z}$ in the support of $\mathscr{D}_{c,\sigma}$.
- The left edges in the DDG tree are labelled with 0, and the right edges in the DDG tree are labelled with 1.
- If for some $x \in \mathbb{Z}$, the k-th bit of $\mathscr{D}_{c,\sigma}(x)$ in binary representation after the decimal point is 1, then there exists exactly one leaf labelled with x at the level k, i.e. $\sum_{k=0}^{m} \frac{c_{x,k}}{2^k} = \mathscr{D}_{c,\sigma}(x)$, where $c_{x,k} \in \{0,1\}$ is the number of leaves labelled with x at the level k.

Figure 3.1: DDG tree, figure adapted from [RVV13].



To generate a sample, the Knuth-Yao algorithm randomly traverses the DDG tree: starting from the root, the algorithm goes along the labelled edges by consuming one random bit per edge, until the algorithm reaches a leaf node. The value on that leaf node is the generated sample.

This algorithm has low bit cost (i.e. the number of random bits consumed for generating a new sample), approximately $4.05+\log_2 \sigma$ on average [KY76]. In addition, the time complexity of the algorithm is similar to its bit cost $\mathcal{O}(\log_2 \sigma)$, because the sampler consumes exactly one random bit during each iteration. A more compact hardware implementation [RVV13] further reduced the memory consumption of this algorithm by partially constructing the DDG tree during the run-time at the expense of a slight overhead on the speed.

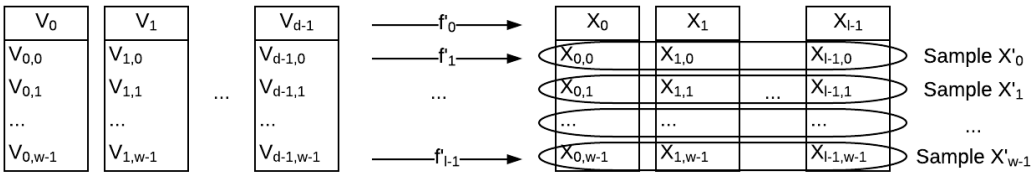
However, both the naive implementation and the compact sampler from [RVV13] are not constant time, since the algorithm may reach a leaf node at some level less than the depth d of the tree. Similar to the constant time CDT method, a naive approach performs a constant time full tree traverse or full table access, at the expense of a much larger overhead compared to the CDT alternatives for the same standard deviation [KRR+18]. Instead, since each unique traverse route on the DDG tree is mapped to a specific sample, Karmakar et al. suggested using the bitslicing method to convert the mapping between a unique bit stream of length d representing a traverse route and the binary bits in the associated sample on the leaf node, to a sequence of boolean expressions [KRR+18]. To accelerate this approach, they performed the bit operations on vectors instead and employed Single Instruction Multiple Data (SIMD) instructions such as the AVX2 to output multiple samples in parallel.

We show the vectorisation approach from [KRR+18] here. Let denote the mapping between the random bit stream $\mathbf{r} \in \{0,1\}^d$ and the binary-represented sample $\mathbf{x} \in \{0,1\}^\ell$ as $x_i = f_i(r_0, \ldots, r_{d-1})$, $0 \leq i \leq \ell - 1$, where x_i and r_i are the i-th coordinate (bit) of $\mathbf{x}$ and $\mathbf{r}$, respectively, and f_i is the i-th mapping function. For d vectors $\mathbf{v}_i \in \{0,1\}^w$ with the length of the machine word width w, to generate w samples in parallel, the algorithm first computes $\mathbf{x}_i = f_i'(\mathbf{v}_0, \ldots, \mathbf{v}_{d-1}) \in \{0,1\}^w$, $0 \leq i \leq \ell - 1$, where f_i' is the SIMD-parallelised variant of f_i for w parallel bits. Then, the sample $\mathbf{x}_j'$ in binary representation is $\mathbf{x}_j' = (x_{0,j}, \ldots, x_{l-1,j}) \in \{0,1\}^\ell$, $0 \leq j \leq w - 1$, where $x_{i,j}$ is the j-th coordinate (bit) of $\mathbf{x}_i$. To optimise the bitslicing circuit, more recently, Karmakar et al. suggested building and connecting small circuit blocks for the input bit streams $\mathbf{r}$ with the same leading bits and using the circuit minimisation heuristics to minimise each circuit block [KRVV19]. They also demonstrated that their bitslicing approach is practical in one of the NIST PQC 3rd round digital signature candidate Falcon [PFH+17].

3.1.3 Rejection & Binary Sampling

The classic rejection sampling algorithm [DN12, von51] can be adapted to sample from a discrete Gaussian distribution. To sample from $\mathcal{D}_{c,\sigma}$, one can sample $x \hookleftarrow \mathcal{U}([c - \tau\sigma, c + \tau\sigma] \cap \mathbb{Z})$ and accept x with probability $\rho_{c,\sigma}(x)$ as the output, where τ is the tail-cut

Figure 3.2: Bitslicing, figure adapted from [KRR+18].



factor (typically, about 10–12). However, this method is slow as the number of trials is $2\tau/\sqrt{2\pi}$ on average (about 8–10 for typical τ). Recently, Karney presented an algorithm sampling exactly from $\mathscr{D}_{c,\sigma}$ without floating-point arithmetic [Kar16], which also has a lower rejection rate compared to the classic rejection sampling algorithm. However, this algorithm needs high number of bit operations to satisfy the precision requirements in cryptography applications. In addition, the original algorithm from [Kar16] is not constant-time. If one attempts to implement the scheme in constant-time, the implementation might be inefficient, since the implementation may always need to perform the worst-case (i.e. maximum) number of bit operations sufficient to the precision requirement of the target cryptography application. More recently, Du et al. extended Karney's algorithm to support σ and c with arbitrary precision [DFW20] by combining a variant of Karney's sampler with the convolution technique from [MW17].

To reduce the rejection rate, recent works performed rejection sampling with regards to some distributions much closer to $\mathscr{D}_{c,\sigma}$ compared to a uniform distribution. The Falcon signature [PFH+17] and its constant-time variant [PRR19] adapted a rejection sampling method with regards to bimodal Gaussians: to sample from $\mathscr{D}_{c,\sigma}$ where $c \in [0,1]$, one can choose some $\sigma' \geq \sigma$ and sample $x \hookleftarrow \mathscr{D}^+_{\sigma'}$ (i.e. the discrete Gaussian distribution $\mathscr{D}_{\sigma'}$ restricted to the domain $\mathbb{Z}^+ \cup \{0\}$). The algorithm computes $x' = b + (2b-1) \cdot x$ where $b \hookleftarrow \mathscr{U}(\{0,1\})$. It was proved by [PFH+17, PRR19] that x' has a bimodal Gaussian distribution close to the target distribution. The algorithm then accepts x' with probability $C(\sigma) \cdot \exp\left(\frac{x^2}{2\sigma'^2} - \frac{(x'-c)^2}{2\sigma^2}\right)$ as the output, where the scaling factor $C(\sigma) = \min(\sigma)/\sigma$ when sampling from multiple σ. This scheme has the average acceptance rate $C(\sigma) \cdot \rho_{c,\sigma}(\mathbb{Z})/(2\rho_{\sigma'}(\mathbb{Z}^+ \cup \{0\}))$, which is proportional to $\min(\sigma)/\sigma'$ [PFH+17, PRR19]. However, if the application needs to sample from different σ, the acceptance probability is high only when $\min(\sigma)$ and $\max(\sigma)$ are sufficiently close. This is not an issue in the Falcon signature, since the parameters in Falcon implies σ' is very close to $\max(\sigma)$ and $\min(\sigma)/\max(\sigma) \approx 0.73$ [PRR19]. However, if the gap between $\min(\sigma)$ and $\max(\sigma)$ is large, since $\sigma' \geq \max(\sigma)$, this algorithm might have a low acceptance rate.

3.1.3.1 Binary Sampling Method

For the zero-centered discrete Gaussian distribution $\mathscr{D}_\sigma$, another rejection sampling algorithm that performs the rejection sampling with regards to some distributions much closer to $\mathscr{D}_\sigma$ compared to a uniform distribution is the binary sampling method used in the BLISS signature scheme [DDLL13]. Let $\sigma = k\sigma_0$, $k \in \mathbb{Z}^+$, and $\sigma_0 = \sqrt{1/(2\ln 2)}$. This algorithm samples from $\mathscr{D}_\sigma^+$ by first generating a sample $x \hookleftarrow \mathscr{D}_{\sigma_0}^+$ from the base sampler and an integer $y \hookleftarrow \mathscr{U}(\{0,\ldots,k-1\})$, then performing a rejection sampling on $z = kx + y$, with the acceptance rate:

$$p = \exp\left(\frac{-y(y+2kx)}{2\sigma^2}\right). \tag{3.1}$$

To generate negative samples, one can sample and apply a random sign bit, with the exception of the rejection with probability $1/2$ when $z = 0$.

Theorem 8 (Adapted from [DDLL13], Thm. 6.6)**.** *Given $x \hookleftarrow \mathscr{D}_{\sigma_0}^+$ and $y \hookleftarrow \mathscr{U}(\{0,\ldots,k-1\})$, the probability to output some integer $z = kx + y$ is proportional to:*

$$\rho_{\sigma_0}(x) \cdot p = \exp\left(-\frac{x^2}{2\sigma_0^2} - \frac{-y(y+2kx)}{2(k\sigma_0)^2}\right) = \exp\left(-\frac{(kx+y)^2}{2(k\sigma_0)^2}\right) = \rho_{k\sigma_0}(z) = \rho_\sigma(x).$$

The rejection framework of the binary sampling algorithm is shown in Algorithm 3.1. The rejection sampling itself will not leak any secret information, if the underlying base sampler and the Bernoulli sampler are side-channel resistant. Unfortunately, to achieve efficient algorithms, the original sampler implementations in the BLISS signature are not constant-time (see Algorithm 3.2 and Algorithm 3.3, respectively). When attacking signature schemes similar to the BLISS, the attacker can gather the discrete Gaussian vectors, or the intermediate base samples and Bernoulli samples, by exploiting the side-channels, such as the cache [BHLY16], or timing and power [EFGT17], and then recover the signing key by using the leaked information. These attacks only require about several thousand signatures and the corresponding samples to succeed.

To mitigate these side-channel attacks against the binary sampling method, several efforts have been proposed. We review them now.

Random Shuffle One commonly used heuristic countermeasure is performing the Fisher-Yates random shuffle (or Knuth shuffle) [Knu98], to mask the relation between the retrieved side-channel information of the samples and the secret, after performing non-constant time sampling schemes [RRVV14, Saa16]. However, in the above mentioned attacking scenarios against the digital signature schemes, the random permutation cannot totally hide the statistical features of the distributions in the attacked vector. By performing statistical analysis, it was shown by [Pes16] that an attacker only requires

Algorithm 3.1 Binary sampling scheme [DDLL13].

Output: A sample from $\mathscr{D}_{\sigma}^{+}$.

function BinarySampler(k)
  Let $x \hookleftarrow \mathscr{D}_{\sigma_0}^{+}$.
  Let $y \hookleftarrow \mathscr{U}(\{0, \dots, k-1\})$.
  Let $z = kx + y$.
  Let $t = y(y + 2kx)$.
  Let $b \hookleftarrow \mathscr{B}_{\exp(-t/(2\sigma^2))}$.
  if $b = 0$ **then**
    Restart BinarySampler.
  end if
  return z.
end function

Algorithm 3.2 Base sampler from BLISS [DDLL13].

Output: A sample from $\mathscr{D}_{\sigma_0}^{+}$.

function BaseSampler
  Sample $b \hookleftarrow \mathscr{U}(\{0, 1\})$.
  if $b = 0$ **then**
    return 0.
  end if
  $i = 1$
  while true do
    Sample $(b_1, b_2, \dots, b_{2i-1}) \hookleftarrow (\mathscr{U}(\{0, 1\}))^{2i-1}$.
    if $(b_1, b_2, \dots, b_{2i-2}) \neq (0, 0, \dots, 0)$ **then**
      Restart BaseSampler.
    end if
    if $b_{2i-1} = 0$ **then**
      return i.
    end if
    $i = i + 1$.
  end while
end function

Algorithm 3.3 Bernoulli sampler from BLISS [DDLL13].

Input: Integer $t = y(y + 2kx)$ with $0 \leq t < 2^{\ell}$ and binary form $t = t_{\ell-1} \dots t_0$, where $x \hookleftarrow \mathscr{D}^{+}_{\sigma_0}$ and $y \hookleftarrow \mathscr{U}(\{0, \dots, k-1\})$. Pre-computed table $p_i = \exp(-2^i/(2\sigma^2))$ for $i < \ell$.

Output: A sample from $\mathscr{B}_p$, where $p = \exp(-t/(2\sigma^2))$.

```
1:  function BernoulliSampler(t)
2:      for i = ℓ − 1 downto 0 do
3:          if t_i = 1 then
4:              Sample a ↩ B_{p_i}.
5:              if a = 0 then
6:                  return 0.
7:              end if
8:          end if
9:      end for
10:     return 1.
11: end function
```

marginally larger yet still practical number of samples to rearrange the coordinates and "undo" the shuffle.

Constant-time Base/Bernoulli Sampler The base sampler of the binary sampling method can be implemented in constant-time by using a full-table access CDT sampler [BCNS15]. More recently, Howe et al. suggested using a binary search CDT sampler with constant number of iterations $\mathcal{O}(\log_2 B)$ on hardware [HKR+18], where B is the tail-cut bound. However, the memory access in this approach is not constant, which might cause potential cache timing leakage in software implementations [KRR+18]. On the other hand, for the table-based Bernoulli sampler, the countermeasure of removing the branches and performing full-table access (see Algorithm 3.4) was suggested by [BHLY16, PBY17, EFGT17]. However, this countermeasure adds significant overhead, since it requires additional randomness for each table entry. A more recent lattice-based digital signature implementation in the NIST PQC 1st round submission, qTesla-R1 [BAA+17], suggested a more efficient approach that the sampler computes the bias p in Equation 3.1 by multiplying table entries from each sub-table based on the binary representation of the input, where every sub-table $\mathscr{B}_i$ has $32 \cdot 8 = 256$ bytes (see Algorithm 3.5). However, although the number of iterations in this sampler is constant, the memory access pattern depends on the size of the underlying CPU cachelines. This could cause a potential leakage via cache timing side-channels on some architectures.

On the other hand, Du et al. suggested that the binary sampling method can also be extended to the arbitrary-centered discrete Gaussian distributions [DWZ19]. For any center $c \in \mathbb{R}$, sampling from $\mathscr{D}_{c,\sigma}$ is equivalent to sampling from $\mathscr{D}_{c_F,\sigma} + \lfloor c \rceil$, where $c_F = c - \lfloor c \rceil \in [0, 1)$ is the fractional part of c. In addition, for $c_F \in [1/2, 1)$, sampling

Algorithm 3.4 Constant-time Bernoulli sampler [PBY17, EFGT17].

Input: Integer $t = y(y + 2kx)$ with $0 \leq t < 2^{\ell}$ and binary form $t = t_{\ell-1} \dots t_0$, where $x \hookleftarrow \mathcal{D}^{+}_{\sigma_0}$ and $y \hookleftarrow \mathcal{U}(\{0, \dots, k-1\})$. Pre-computed table $p_i = \exp(-2^i/(2\sigma^2))$ for $i < \ell$.

Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/(2\sigma^2))$.

1: **function** BernoulliSampler(t)
2: Let $r = 1$.
3: **for** $i = \ell - 1$ **downto** 0 **do**
4: Sample $a \hookleftarrow \mathcal{B}_{p_i}$.
5: Set $r = r \cdot (1 - t_i + at_i)$.
6: **end for**
7: **return** r.
8: **end function**

Algorithm 3.5 Bernoulli sampler with constant number of iterations [BAA+17].

Input: Integer $t = y(y+2kx)$ with $0 \leq t < 2^{15}$, where $x \hookleftarrow \mathcal{D}^{+}_{\sigma_0}$ and $y \hookleftarrow \mathcal{U}(\{0, \dots, k-1\})$. Pre-computed Bernoulli table entries $\mathcal{B}_{i,j} = \exp(-2^{(j \cdot 32^i)}/(2\sigma^2))$, where $i, j \in \mathbb{Z}^+$, $0 \leq i < 3$, and $0 \leq j < 32$. Each $\mathcal{B}_{i,j}$ has 8 bytes.

Output: A sample from $\mathcal{B}_p$, where $p = \exp(-t/(2\sigma^2))$.

1: **function** BernoulliSampler(t)
2: Sample $r \hookleftarrow \mathcal{U}(\{0,1\}^{62})$.
3: Let $c = 2^{62}$.
4: Let $s = t$.
5: **for** $i = 0$ **to** 2 **do**
6: Set $c = c \cdot \mathcal{B}_{i,s \bmod 32}$.
7: Set $s = s/32$.
8: **end for**
9: **if** $r \geq \lfloor c \rfloor$ **then**
10: **return** 0.
11: **else**
12: **return** 1.
13: **end if**
14: **end function**

from $\mathcal{D}_{c_F,\sigma}$ is equivalent to sampling from $1 - \mathcal{D}_{c'_F,\sigma}$ where $c'_F = 1 - c_F \in (0, 1/2]$. A modified binary sampling scheme [DWZ19] (see Algorithm 3.6) can then be adapted to sample from $\mathcal{D}_{c'_F,\sigma}$ with any $c'_F \in (0, 1/2]$, in which the average number of trials is upper-bounded by $\frac{\sigma^2}{\sigma_0\sigma - \sigma_0^2} \cdot \frac{\rho_{\sigma_0}(\mathbb{Z}^+)}{\sigma\sqrt{\pi/2} - 1}$, where $\sigma_0 = \sqrt{1/(2\ln 2)}$ and $\sigma = k\sigma_0$ for some $k \in \mathbb{Z}^+$. This upper-bound is about 1.47 for large σ [DWZ19], which is the same as the original binary sampling method [DDLL13].

Algorithm 3.6 Non-zero centered binary sampling scheme [DWZ19].

Input: Center $c'_F \in (0, 1/2]$.
Output: A sample from $\mathcal{D}_{c'_F,\sigma}$.

function BinarySamplerEx(c'_F, k)
  Let $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$.
  Let $y \hookleftarrow \mathcal{U}(\{0, \ldots, k-1\})$.
  Let $s \hookleftarrow \mathcal{U}(\{-1, 1\})$.
  Let $\delta = \lceil kx + sc'_F \rceil - kx - sc'_F$.
  if $y + \delta \geq k$ **then**
    Restart BinarySamplerEx.
  end if
  Let $z = \lceil kx + sc'_F \rceil + y$.
  Let $t = 2kx(y + \delta) + (y + \delta)^2$.
  Let $b \hookleftarrow \mathcal{B}_{\exp(-t/(2\sigma^2))}$.
  if $b = 0$ **then**
    Restart BinarySamplerEx.
  end if
  return $s \cdot z$.
end function

3.1.4 Convolution Methods

For the zero-centered discrete Gaussian distributions, one can apply the following KLD-based convolution theorem [PDG14, KHR+18] to construct discrete Gaussian sampling algorithms:

Theorem 9 (KLD-based Convolution Theorem, Adapted from [PDG14], Lemma 3)**.** *Let $x_1 \hookleftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}$ and $x_2 \hookleftarrow \mathcal{D}_{k\mathbb{Z},\sigma_2}$ for some $\sigma_1, \sigma_2 \in \mathbb{R}^+$. Let $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$. For any $\epsilon \in (0, 1/2)$, if $\sigma_1 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$ and $\sigma_3 \geq \eta_\epsilon(k\mathbb{Z})/\sqrt{2\pi}$, then the distribution $\mathcal{P}$ of $x_1 + x_2$ satisfies:*

$$KL(\mathcal{P} \| \mathcal{D}_\sigma) \leq 2\left(1 - \left(\frac{1+\epsilon}{1-\epsilon}\right)^2\right)^2 \approx 32\epsilon^2.$$

For the deviation $\sigma \approx 215$ in the BLISS-I parameter set [DDLL13], one can generate $x_1, x_2 \hookleftarrow \mathcal{D}_{\sigma_1}$ and compute $x_1 + k_1 x_2$, where $\sigma_1 = \sigma/\sqrt{1 + k_1^2} \approx 19.53$ and $k_1 = 11$. Sampling from $\mathcal{D}_{\sigma_1}$ can be further decomposed into $x_3 + k_2 x_4$, where $x_3, x_4 \hookleftarrow \mathcal{D}_{\sigma_2}$, $\sigma_2 =$

$\sigma_1/\sqrt{1+k_2^2} \approx 6.18$, and $k_2 = 3$ (see Algorithm 3.7). If the sampling algorithm of $\mathscr{D}_{\sigma_2}$ (or $\mathscr{D}_{\sigma_1}$) is constant-time, then the whole sampling scheme will be constant-time. To sample from $\mathscr{D}_{\sigma_2}$, Karmakar et al. adapted the bitslicing method to implement the Knuth-Yao algorithm [KY76] more efficiently in constant-time [KRR+18, KRVV19], compared to the previous full-table access CDT approach.

Algorithm 3.7 KLD-based convolution sampling scheme [PDG14, KHR+18].

Output: A sample from $\mathscr{D}_\sigma$, where $\sigma \approx 215$.

1: **function** ConvolutionSampler
2: Sample $x_1, x_2, x_3, x_4 \hookleftarrow \mathscr{D}_{\sigma_0}$, where $\sigma_0 \approx 6.18$.
3: Let $y = (x_1 + 3x_2) + 11 \cdot (x_3 + 3x_4)$.
4: **return** y.
5: **end function**

Meanwhile, Micciancio and Walter proposed the following max-log based convolution theorems [MW17]:

Theorem 10 (Adapted from [MW17], Cor. 4.1). *Let $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{Z}^n$ be a nonzero vector with $\gcd(z_1, \ldots, z_n) = 1$ and $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n$ with $\sigma_i \geq \|\mathbf{z}\|_\infty \cdot \eta_\epsilon(\mathbb{Z})/\sqrt{\pi}$ for all $i \leq n$. Let $\mathbf{y} \hookleftarrow (\mathscr{D}'_{\sigma_i})^n$, with $ML(\mathscr{D}'_{\sigma_i} \| \mathscr{D}_{\sigma_i}) \leq \mu_i$ for all i. Let $\sigma^2 = \sum z_i^2 \sigma_i^2$ and $\mathscr{P}$ be the distribution of $\sum z_i y_i$. Then, $ML(\mathscr{P} \| \mathscr{D}_\sigma) \leq 2\epsilon + \sum \mu_i$.*

Theorem 11 (Adapted from [MW17], Cor. 4.2). *Let $x_1 \hookleftarrow \mathscr{D}'_{\mathbb{Z},\sigma_1}$ and $x_2 \hookleftarrow \mathscr{D}'_{k\mathbb{Z},\sigma_2}$ for some $\sigma_1, \sigma_2 \in \mathbb{R}^+$. Let $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$. If $\sigma_1 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$, $\sigma_3 \geq \eta_\epsilon(k\mathbb{Z})/\sqrt{2\pi}$, $ML(\mathscr{D}'_{\mathbb{Z},\sigma_1} \| \mathscr{D}_{\mathbb{Z},\sigma_1}) \leq \mu_1$, and $ML(\mathscr{D}'_{k\mathbb{Z},\sigma_2} \| \mathscr{D}_{k\mathbb{Z},\sigma_2}) \leq \mu_2$, then the distribution $\mathscr{P}$ of $x_1 + x_2$ satisfies $ML(\mathscr{P} \| \mathscr{D}_\sigma) \leq 4\epsilon + \mu_1 + \mu_2$.*

For the arbitrary-centered discrete Gaussian distributions, Micciancio and Walter presented a recursive convolution sampling scheme for $\mathscr{D}_{c,\sigma}$ [MW17] as follows: suppose the center c has k fractional bits. Let $\sigma_0 = \sigma/\sqrt{\sum_{i=0}^{k-1} 2^{-2i}}$. One can sample $x_k \hookleftarrow \mathscr{D}_{c_k,\sigma_0}$ where $c_k = 2^{k-1} \cdot c$, then use $y_k = 2^{-k+1} \cdot x_k$ to round c to a new center $c' = c - y_k$ with $k' = k-1$ fractional bits. Set $c = c'$ and $k = k'$ in the next iteration until $k = 0$, and $\sum_{i=1}^{k} y_i$ will be a sample distributed as $\mathscr{D}_{c,\sigma}$. The authors separated this algorithm into an online phase and an offline phase, where the offline phase will generate samples x_i in batch and the online phase will compute the linear combinations of x_i for $i \in \{1, \ldots, k\}$ [MW17]. The online phase is very fast and can be implemented in constant-time. However, for implementations where both sampling from $\mathscr{D}_{c_i,\sigma_0}$ and computing the linear combinations need to be carried during the run-time, it is unclear how to efficiently implement the $\mathscr{D}_{c_i,\sigma_0}$ sampling algorithm in constant-time (which is another discrete Gaussian sampler supporting a small amount of centers c_i). The offline batch sampler also consumes significant amount of memory.

3.2 Lattice-based (Hierarchical) Identity-based Encryption

In this section, we focus on the existing *practical* implementations of lattice-based (hierarchical) identity-based encryption schemes. Typically, the lattice-based (H)IBE constructions rely on the lattice trapdoors [GPV08, ABB10a]. Two types of lattice trapdoors have been employed in lattice-based (H)IBE schemes with practical implementation results: the (Mod)NTRU trapdoor [HHP+03, SS11, CKKS19] and the gadget trapdoor [MP12, GM18].

(Mod)NTRU Trapdoor Based IBE Although the theoretical construction of lattice-based (H)IBE was discussed by [GPV08, ABB10a], however, it was never instantiated in practice until 2014 that Ducas et al. proposed the first efficient lattice-based identity-based encryption scheme with implementation results [DLP14]. They based their construction on the IBE scheme by [GPV08], using a variant of NTRU lattices [SS11]. The underlying security problems are the NTRU problem [HHP+03, SS11] for key generation and Ring Learning with Errors (Ring-LWE) [LPR10] for the encryption. They showed that the NTRU trapdoor is very efficient to sample in software implementation. The use of structured lattices also allowed for implementation optimisations such as the Number Theoretic Transform (NTT), as demonstrated later by McCarthy et al., whose software performance of the DLP IBE [MSO17] outperformed that of a classical elliptic-curve based IBE scheme [BF03]. McCarthy et al. also showed that 64-bit precision is sufficient to implement the integer discrete Gaussian sampling subroutine used by the lattice discrete Gaussian sampler [Kle00, GPV08] for the DLP IBE parameter sets [MSO17]. More recently, Cheon et al. provided an IBE scheme [CKKS19] based on the ModNTRU lattice [CKKS19, CPS+20], which is essentially the generalisation of the DLP IBE scheme [DLP14] to ModNTRU lattices with lattice dimension greater than $2N$.

Gadget Trapdoor Based IBE Another type of lattice trapdoors which is efficient to sample in practice is the gadget trapdoor [MP12, GM18]. One advantage of the gadget trapdoor is that it is based on the standard assumptions i.e. does not require the NTRU assumptions [HHP+03, SS11]. Bert et al. proposed the first practical lattice-based IBE scheme based on gadget trapdoor with implementation result [BFRS18]. They built their construction on the Ring-LWE version of the IBE scheme based on standard models [ABB10a], using a variant of gadget trapdoors [GM18]. This is also the first practical lattice-based IBE implementation based on standard models, since the DLP IBE scheme [DLP14] is based on the random oracle models. The authors also showed that double precision floating-point arithmetic is sufficient for their parameter sets [BFRS18], by

directly adapting the precision analysis results from [GM18]. Very recently, Bert et al. [BEP+21] proposed an IBE scheme based on the module [LS15] gadget trapdoor, which is essentially the module lattice version of the IBE scheme from [BFRS18]. The authors also introduced new lattice discrete Gaussian sampling techniques to module gadget lattice settings. However, the arithmetic precision requirement of the lattice discrete Gaussian sampler in [BEP+21] is unclear, since it was not discussed in details by the authors.

For lattice-based *hierarchical* identity-based encryption schemes, although there exists several theoretical constructions [CHKP10, ABB10a, ABB10b], however, to the best of our knowledge, the only practical implementation result of lattice-based HIBE by far is the *partial* evaluation result of the Latte HIBE [ETS19]. The Latte HIBE [ETS19] can be considered as a combination of the DLP IBE scheme [DLP14] with the Bonsai-tree HIBE construction [CHKP10] to create a hierarchical lattice-based IBE scheme. However, the proposed Latte specification [ETS19] only provided the Encrypt and Decrypt performance results, and it was unclear if Latte KeyGen, Delegate, and Extract are practical at all.

Here, we summarise the KeyGen, Delegate, Extract, Encrypt, and Decrypt algorithms of Latte [ETS19].

3.2.1 Summary of Latte HIBE Scheme

The Latte parameter sets are shown in Table 3.1. In the following Latte algorithm description, user identities at level ℓ are denoted ID_ℓ. A hash function from an arbitrary length input to a vector of integers of length N is written as $H : \{0,1\}^* \to \mathbb{Z}_q^N$.

Table 3.1: Latte Parameters [ETS19].

Set	**Security**	N	q	σ_ℓ		
				$\ell = 0$	$\ell = 1$	$\ell = 2$
Latte-1	128	1024	$2^{24} - 2^{14} + 1$	105.9	5499.6	-
Latte-2	256	2048	$2^{25} - 2^{12} + 1$	105.9	7880.6	-
Latte-3	80	1024	$2^{36} - 2^{20} + 1$	6777.4	351958.7	22559368.5
Latte-4	160	2048	$2^{38} - 2^{26} + 1$	9583.5	713152.4	65487839.3

KeyGen The Latte KeyGen algorithm is shown in Algorithm 3.8. The algorithm is very similar to the KeyGen algorithm of the DLP IBE scheme [DLP14], which generates a secret short basis $\mathbf{S}_0 = \begin{pmatrix} \mathcal{A}(\mathbf{g}) & -\mathcal{A}(\mathbf{f}) \\ \mathcal{A}(\mathbf{G}) & -\mathcal{A}(\mathbf{F}) \end{pmatrix}$ and the associated public basis $\mathbf{B}_0 = \begin{pmatrix} -\mathcal{A}(\mathbf{h}) & \mathbf{I}_N \\ q\mathbf{I}_N & \mathbf{0}_N \end{pmatrix}$ for some $\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G} \in \mathfrak{R}$ such that $\mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$ and $\mathbf{h} = \mathbf{g}/\mathbf{f} \mod q$ (see Section 2.2.1). The algorithm first samples $\mathbf{f}, \mathbf{g}$ from $\mathscr{D}_{\sigma_0}$, where $\sigma_0 = 1.17\sqrt{q/(2N)}$ [DLP14]. Then, the KeyGen algorithm performs the norm check (Line 3–6 in Algorithm 3.8) to ensure the Gram-Schmidt norm $\|\tilde{\mathbf{S}}_0\|$ of the generated NTRU basis $\mathbf{S}_0$ is less than $\sigma_0\sqrt{2N}$.

This is necessary since in order to use the Klein-GPV sampler to sample from $\mathscr{D}_{\Lambda(\mathbf{S}_{\ell-1}),\mathbf{c},\sigma_\ell}$ in Latte Delegate and Extract (Line 6 in Algorithm 3.9; Line 5 in Algorithm 3.10), σ_ℓ must be greater than $(\eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}) \cdot \|\tilde{\mathbf{S}}_{\ell-1}\|$ for $\ell > 0$ and some $\eta_\epsilon(\mathbb{Z})$ [GPV08, DN12], where $\sigma_\ell = (\eta_\epsilon(\mathbb{Z})/\sqrt{2\pi})\sqrt{(\ell+1)N}\sigma_{\ell-1}$ in the Latte parameter sets [ETS19]. Then, the algorithm uses the resultant method [HHP+03, DLP14] to find $\mathbf{F}', \mathbf{G}' \in \mathfrak{R}$ such that $\mathbf{fG}' - \mathbf{gF}' = q \mod x^N + 1$ (Line 7–12 in Algorithm 3.8). The coefficient size of $\mathbf{F}', \mathbf{G}'$ is then reduced by length reduction [HHP+03, DLP14] (Line 13–14 in Algorithm 3.8). The master private key at level 0 is $\mathbf{S}_0 = \begin{pmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{pmatrix} \in \mathfrak{R}^{2\times 2}$, and the master public key at level 0 is $(\mathbf{h}, \mathbf{b}) \in \mathfrak{R}_q^2$ for $\mathbf{h} = \mathbf{g}/\mathbf{f} \mod q$ and some random $\mathbf{b} \hookleftarrow \mathscr{U}(\mathfrak{R}_q)$.

Algorithm 3.8 Latte KeyGen algorithm [ETS19].

Input: N, q, σ_0.
Output: $\mathbf{S}_0 \in \mathfrak{R}^{2\times 2}, \mathbf{h}, \mathbf{b} \in \mathfrak{R}_q$.

1: **function** KeyGen
2: $\mathbf{f}, \mathbf{g} \hookleftarrow \mathscr{D}_{\sigma_0}^N$.
3: Norm $\leftarrow \max\left(\|\mathbf{g}, -\mathbf{f}\|, \left\|\left(\frac{q\cdot\mathbf{f}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}, \frac{q\cdot\mathbf{g}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}\right)\right\|\right)$.
4: **if** Norm $> \sigma_0 \cdot \sqrt{2N}$ **then**
5: **goto** Step 2.
6: **end if**
7: Using extended Euclidean algorithm, compute $\mathbf{u}_f, \mathbf{u}_g \in \mathfrak{R}$ and $r_f, r_g \in \mathbb{Z}$ such that $\mathbf{f}\cdot\mathbf{u}_f = r_f \mod x^N + 1$ and $\mathbf{g}\cdot\mathbf{u}_g = r_g \mod x^N + 1$.
8: **if** $r_f = 0 \mod q$ **or** $\gcd(r_f, r_g) > 1$ **then**
9: **goto** Step 2.
10: **end if**
11: Using extended Euclidean algorithm, compute $v_f, v_g \in \mathbb{Z}$ such that $v_f r_f + v_g r_g = 1$.
12: $\mathbf{F}' \leftarrow -qv_g \cdot \mathbf{u}_g$, $\mathbf{G}' \leftarrow qv_f \cdot \mathbf{u}_f$.
13: $\mathbf{k} = \left\lfloor \frac{\mathbf{F}'\cdot\mathbf{f}^*+\mathbf{G}'\cdot\mathbf{g}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*} \right\rceil \in \mathfrak{R}$.
14: $\mathbf{F} \leftarrow \mathbf{F}' - \mathbf{k}\cdot\mathbf{f}$, $\mathbf{G} \leftarrow \mathbf{G}' - \mathbf{k}\cdot\mathbf{g}$.
15: $\mathbf{h} \leftarrow \mathbf{g}\cdot\mathbf{f}^{-1} \mod q$.
16: $\mathbf{b} \hookleftarrow \mathscr{U}(\mathfrak{R}_q)$.
17: **return** $\mathbf{S}_0 = \begin{pmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{pmatrix}, \mathbf{h}, \mathbf{b}$.
18: **end function**

Delegate The Latte Delegate algorithm is shown in Algorithm 3.9. Let denote $\mathbf{A}_i = H(\mathsf{ID}_1|\ldots|\mathsf{ID}_i)$ and $\mathbf{h}_i = (\mathbf{h}, \mathbf{A}_1, \ldots, \mathbf{A}_i)$ for $1 \le i \le \ell$, where $\mathbf{h}$ is from the master public key at level 0. Assume the master private key $\mathbf{S}_{\ell-1}$ from level $\ell-1$ is a (Mod)NTRU basis associated with $\mathbf{h}_{\ell-1}$. For user identity ID_ℓ at level ℓ, the algorithm generates a secret

short $(\ell+2)N$-dimensional ModNTRU basis:

$$\mathbf{S}_\ell = \begin{pmatrix} \mathcal{A}(\mathbf{s}_{0,0}) & \mathcal{A}(\mathbf{s}_{0,1}) & \dots & \mathcal{A}(\mathbf{s}_{0,d-1}) \\ \mathcal{A}(\mathbf{s}_{1,0}) & \mathcal{A}(\mathbf{s}_{1,1}) & \dots & \mathcal{A}(\mathbf{s}_{1,d-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}(\mathbf{s}_{d-1,0}) & \mathcal{A}(\mathbf{s}_{d-1,1}) & \dots & \mathcal{A}(\mathbf{s}_{d-1,d-1}) \end{pmatrix},$$

associated with $\mathbf{h}_\ell$ (see Section 2.2.1), by using Klein-GPV sampler [Kle00, GPV08] with $\mathbf{S}_{\ell-1}$. At the beginning, the algorithm generates $\ell+1$ short vectors $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{\ell+1}) \in \mathfrak{R}^{\ell+2}$ from the ModNTRU lattice associated with $\mathbf{h}_\ell$ (Line 3–10 in Algorithm 3.9). To realise this, for each vector $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{\ell+1})$, the algorithm first samples $\mathbf{s}_{i,\ell+1} \hookleftarrow \mathcal{D}^N_{\sigma_\ell}$ (Line 4 in Algorithm 3.9). Then, the algorithm samples $\mathbf{z} \hookleftarrow \mathcal{D}_{\Lambda(\mathbf{S}_{\ell-1}),\mathbf{c},\sigma_\ell}$ by using Klein-GPV sampler [Kle00, GPV08] for center $\mathbf{c} = (-\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \dots, \mathbf{0})$ and let $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell}) \leftarrow \mathbf{c}-\mathbf{z}$ (Line 5–6 in Algorithm 3.9). From [GPV08], $\mathbf{c}-\mathbf{z}$ follows the distribution $\mathcal{D}_{\mathbf{c}+\Lambda(\mathbf{S}_{\ell-1}),\sigma_\ell}$. Since $\Lambda(\mathbf{S}_{\ell-1})$ is a (Mod)NTRU lattice, by Definition 2, we have the following equations in $\mathfrak{R}_q$:

$$\begin{aligned} &(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell}) \cdot (1, \mathbf{h}, \mathbf{A}_1, \dots, \mathbf{A}_{\ell-1}) = -\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell \\ \Longrightarrow\ &(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell+1}) \cdot (1, \mathbf{h}, \mathbf{A}_1, \dots, \mathbf{A}_{\ell-1}, \mathbf{A}_\ell) = \mathbf{0}. \end{aligned}$$

Therefore, by Definition 2, $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell+1})$ is in the ModNTRU lattice associated with $\mathbf{h}_\ell$. Then, similar to the Latte KeyGen, the algorithm also performs the norm check (Line 7–9 in Algorithm 3.9) to ensure the Gram-Schmidt norm $\|\tilde{\mathbf{S}}_\ell\|$ of the generated ModNTRU basis $\mathbf{S}_\ell$ is less than $\sigma_\ell\sqrt{(\ell+2)N}$. The remainder of the Delegate algorithm, in which $(\mathbf{s}_{\ell+1,0}, \mathbf{s}_{\ell+1,1}, \dots, \mathbf{s}_{\ell+1,\ell+1})$ is generated (Line 11–25 in Algorithm 3.9), is a higher-dimensional analogue of Latte KeyGen. The algorithm first uses the resultant method to find $\mathbf{s}_{\ell+1,j} \in \mathfrak{R}$, $0 \le j \le \ell+1$, such that $\mathbf{s}_{\ell+1,0} \cdot \mathbf{M}_0 + \dots + \mathbf{s}_{\ell+1,\ell+1} \cdot \mathbf{M}_{\ell+1} = q \bmod x^N+1$, where $\mathbf{M}_j = \mathbf{C}_{\ell+1,j}$ for the cofactor matrix $\mathbf{C}$ of $\mathbf{S}'_\ell = (\mathbf{s}_{i,j})_{0\le i,j\le \ell+1} \in \mathfrak{R}^{(\ell+2)\times(\ell+2)}$ (Line 11–19 in Algorithm 3.9). Therefore, $\det(\mathbf{S}'_\ell) = q$ and $\mathbf{S}_\ell$ is a ModNTRU basis (see Section 2.2.1). Cramer's rule is then utilised to reduce the size of coefficients in $(\mathbf{s}_{\ell+1,0}, \mathbf{s}_{\ell+1,1}, \dots, \mathbf{s}_{\ell+1,\ell+1})$ (Line 20–25 in Algorithm 3.9). The delegated master private key for user identity ID_ℓ at level ℓ is $\mathbf{S}_\ell = (\mathbf{s}_{i,j})_{0\le i,j\le \ell+1} \in \mathfrak{R}^{(\ell+2)\times(\ell+2)}$.

Extract The Latte Extract algorithm is shown in Algorithm 3.10. For user identity ID_ℓ at level ℓ, the algorithm generates a secret short solution $(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_{\ell+1})$ to the equation:

$$\mathbf{t}_0 + \mathbf{t}_1 \cdot \mathbf{h} + \mathbf{t}_2 \cdot \mathbf{A}_1 + \dots + \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell = \mathbf{b} \mod q, \tag{3.2}$$

by using the Klein-GPV sampler [Kle00, GPV08] with the master private key $\mathbf{S}_{\ell-1}$ from level $\ell-1$, where $\mathbf{A}_i = H(\mathsf{ID}_1 | \dots | \mathsf{ID}_i)$ for $1 \le i \le \ell$ and $\mathbf{h}, \mathbf{b}$ are from the master public

Algorithm 3.9 Latte Delegate algorithm (from level $\ell - 1$ to ℓ) [ETS19].

Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathfrak{R}_q, \mathsf{ID}_\ell$.

Output: $\mathbf{S}_\ell \in \mathfrak{R}^{(\ell+2)\times(\ell+2)}$.

function Delegate
  $\mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1|\ldots|\mathsf{ID}_\ell)$.
  for $i = 0$ **to** ℓ **do**
    $\mathbf{s}_{i,\ell+1} \hookleftarrow \mathscr{D}^N_{\sigma_\ell}$.
    $\mathbf{c} \leftarrow (-\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \ldots, \mathbf{0})$.
    $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,\ell}) \leftarrow \mathbf{c} - \text{Klein-GPV}(\mathbf{S}_{\ell-1}, \mathbf{c}, \sigma_\ell)$.
    if $\|(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,\ell}, \mathbf{s}_{i,\ell+1})\| > \sqrt{(\ell+2)N} \cdot \sigma_\ell$ **then**
      Resample.
    end if
  end for
  for $j = 0$ **to** $\ell + 1$ **do**
    $\mathbf{M}_j \leftarrow (-1)^{j+\ell+1} \det \begin{pmatrix} \mathbf{s}_{0,0} & \cdots & \mathbf{s}_{0,j-1} & \mathbf{s}_{0,j+1} & \cdots & \mathbf{s}_{0,\ell+1} \\ \mathbf{s}_{1,0} & \cdots & \mathbf{s}_{1,j-1} & \mathbf{s}_{1,j+1} & \cdots & \mathbf{s}_{1,\ell+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{s}_{\ell,0} & \cdots & \mathbf{s}_{\ell,j-1} & \mathbf{s}_{\ell,j+1} & \cdots & \mathbf{s}_{\ell,\ell+1} \end{pmatrix}$.
    Using extended Euclidean algorithm, compute $\mathbf{u}_j \in \mathfrak{R}$ and $r_j \in \mathbb{Z}$ such that $\mathbf{M}_j \cdot \mathbf{u}_j = r_j \mod x^N + 1$.
  end for
  Using (a series of) extended Euclidean algorithm, compute $v_0, \ldots, v_{\ell+1} \in \mathbb{Z}$ such that $v_0 r_0 + \cdots + v_{\ell+1} r_{\ell+1} = 1$.
  if failed **then**
    goto Step 3.
  end if
  $\mathbf{s}_{\ell+1,j} \leftarrow q v_j \cdot \mathbf{u}_j$, for $0 \le j \le \ell + 1$.
  Set $\mathbf{C} = (\mathbf{c}_{i,j})$, where $\mathbf{c}_{i,j} = \mathbf{s}_{j,0} \cdot \mathbf{s}^*_{i,0} + \cdots + \mathbf{s}_{j,\ell+1} \cdot \mathbf{s}^*_{i,\ell+1}$, $0 \le i, j \le \ell$.
  Set $\mathbf{d} = (\mathbf{d}_i)$, where $\mathbf{d}_i = \mathbf{s}_{\ell+1,0} \cdot \mathbf{s}^*_{i,0} + \cdots + \mathbf{s}_{\ell+1,\ell+1} \cdot \mathbf{s}^*_{i,\ell+1}$, $0 \le i \le \ell$.
  Let $\mathbf{k} = (\mathbf{k}_i)_{0 \le i \le \ell}$ be the solution to $\mathbf{C} \cdot \mathbf{k} = \mathbf{d}$. By Cramer's rule, $\mathbf{k}_i = \frac{\det(\mathbf{C}_i(\mathbf{d}))}{\det(\mathbf{C})}$, where $\mathbf{C}_i(\mathbf{d})$ is the matrix $\mathbf{C}$ with its i^{th} column replaced by $\mathbf{d}$.
  for $i = 0$ **to** ℓ **do**
    $(\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1}) = (\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1}) - \lfloor \mathbf{k}_i \rceil \cdot (\mathbf{s}_{i,0}, \ldots, \mathbf{s}_{i,\ell+1})$.
  end for
  return $\mathbf{S}_\ell = (\mathbf{s}_{i,j})$, for $0 \le i, j \le \ell + 1$.
end function

key at level 0. To realise this, similar to the Latte Delegate, the algorithm first samples $\mathbf{t}_{\ell+1} \hookleftarrow \mathscr{D}_{\sigma_\ell}^N$ (Line 3 in Algorithm 3.10). Then, the algorithm samples $\mathbf{z} \hookleftarrow \mathscr{D}_{\Lambda(\mathbf{S}_{\ell-1}),\mathbf{c},\sigma_\ell}$ by using Klein-GPV sampler [Kle00, GPV08] for center $\mathbf{c} = (\mathbf{b} - \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \dots, \mathbf{0})$ and let $(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_\ell) \leftarrow \mathbf{c} - \mathbf{z}$ (Line 4–5 in Algorithm 3.10). From [GPV08], $\mathbf{c} - \mathbf{z}$ follows the distribution $\mathscr{D}_{\mathbf{c}+\Lambda(\mathbf{S}_{\ell-1}),\sigma_\ell}$. Since $\Lambda(\mathbf{S}_{\ell-1})$ is a (Mod)NTRU lattice, by Definition 2, we have the following equations in $\mathfrak{R}_q$:

$$\begin{aligned}(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_\ell) \cdot (1, \mathbf{h}, \mathbf{A}_1, \dots, \mathbf{A}_{\ell-1}) &= \mathbf{b} - \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell \\ \implies \mathbf{t}_0 + \mathbf{t}_1 \cdot \mathbf{h} + \mathbf{t}_2 \cdot \mathbf{A}_1 + \cdots + \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell &= \mathbf{b}.\end{aligned}$$

Therefore, the user private key for identity ID_ℓ at level ℓ is $(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_{\ell+1}) \in \mathfrak{R}_q^{\ell+2}$.

Algorithm 3.10 Latte Extract algorithm (from level $\ell - 1$ to user at level ℓ) [ETS19].

Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathbb{Z}_q^N, \mathsf{ID}_\ell$.
Output: $\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1} \in \mathfrak{R}_q$.

1: **function** Extract
2: $\quad \mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1 | \dots | \mathsf{ID}_\ell)$.
3: $\quad \mathbf{t}_{\ell+1} \hookleftarrow \mathscr{D}_{\sigma_\ell}^N$.
4: $\quad \mathbf{c} \leftarrow (\mathbf{b} - \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \dots, \mathbf{0})$.
5: $\quad (\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_\ell) \leftarrow \mathbf{c} - \text{Klein-GPV}(\mathbf{S}_{\ell-1}, \mathbf{c}, \sigma_\ell)$.
6: $\quad$ **return** $\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1} \in \mathfrak{R}_q$.
7: **end function**

Encrypt/Decrypt An extended version of traditional Ring-LWE encryption/decryption [LPR10] is used for Latte Encrypt and Decrypt as given in Algorithm 3.11 and Algorithm 3.12, respectively. A random *seed* is sampled and used together with a Key Derivation Function (KDF) to one-time-pad the message μ i.e. $Z \leftarrow \mu \oplus \text{KDF}(seed)$ (Line 2–3 in Algorithm 3.11). The *seed* is encoded to $\mathbf{m} \in \mathfrak{R}_q$ (Line 9 in Algorithm 3.11) and then encrypted using Ring-LWE over $\mathfrak{R}_q$ (Line 5–8, 10 in Algorithm 3.11) [LPR10]:

$$\begin{pmatrix} \mathbf{C}_h \\ \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_\ell \\ \mathbf{C}_b \end{pmatrix} = \begin{pmatrix} \mathbf{h} \\ \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_\ell \\ \mathbf{b} \end{pmatrix} \cdot \mathbf{e} + \begin{pmatrix} \mathbf{e}_h \\ \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_\ell \\ \mathbf{e}_b \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{m} \end{pmatrix},$$

where $\mathbf{e}, \mathbf{e}_1, \dots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b$ are ephemeral private keys deterministically sampled from $\mathscr{D}_{\sigma_e}$ by using the seed $\text{KDF}(seed|Z)$ (Line 4 in Algorithm 3.11), $\mathbf{A}_i = H(\mathsf{ID}_1 | \dots | \mathsf{ID}_i)$ for $1 \leq i \leq \ell$, and $\mathbf{h}, \mathbf{b}$ are from the master public key at level 0. The ciphertext consists of the encrypted message Z, deterministically generated ephemeral public keys $\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h$ and the encrypted *seed*, $\mathbf{C}_b$. This is a variant of the Fujisaki-Okamoto transform [FO99]

to protect against invalid ciphertexts. The Decrypt process takes the user private key $(\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{\ell+1})$ to decrypt the *seed* (Line 2–3 in Algorithm 3.12) and reconstruct the message. This works as follows by operations over $\mathfrak{R}_q$:

$$\begin{aligned}\mathbf{V} &= \mathbf{C}_b - \mathbf{t}_1 \cdot \mathbf{C}_h - \mathbf{t}_2 \cdot \mathbf{C}_1 - \cdots - \mathbf{t}_{\ell+1} \cdot \mathbf{C}_\ell \\ &= (\mathbf{b} \cdot \mathbf{e} + \mathbf{e}_b + \mathbf{m}) - \mathbf{t}_1(\mathbf{h} \cdot \mathbf{e} + \mathbf{e}_h) - \mathbf{t}_2(\mathbf{A}_1 \cdot \mathbf{e} + \mathbf{e}_1) - \cdots - \mathbf{t}_{\ell+1}(\mathbf{A}_\ell \cdot \mathbf{e} + \mathbf{e}_\ell) \\ &= \mathbf{e}_b + \mathbf{m} - \mathbf{t}_1 \cdot \mathbf{e}_h - \mathbf{t}_2 \cdot \mathbf{e}_1 - \cdots - \mathbf{t}_{\ell+1} \cdot \mathbf{e}_\ell + \mathbf{t}_0 \cdot \mathbf{e},\end{aligned}$$

where the first equality is derived by substituting $\mathbf{C}_b, \mathbf{C}_h, \mathbf{C}_i$ with their definitions, and the second equality holds based on Equation 3.2. By construction, the error and private key terms are small enough so that $\mathbf{V}$ is decoded successfully to recover the *seed* (Line 3 in Algorithm 3.12). By using the *seed*, the message μ is straightforwardly recovered from Z (Line 12 in Algorithm 3.12).

Algorithm 3.11 Latte Encrypt algorithm (at level ℓ) [ETS19].

Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \mathrm{KDF}, \mathsf{ID}_\ell, \mu \in \{0,1\}^{256}$.
Output: $Z \in \{0,1\}^{256}, \mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathfrak{R}_q$.

1: **function** Encrypt
2: $\quad$ *seed* $\hookleftarrow \{0,1\}^{256}$.
3: $\quad$ $Z \leftarrow \mu \oplus \mathrm{KDF}(\textit{seed})$.
4: $\quad$ $\mathbf{e}, \mathbf{e}_1, \ldots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b \hookleftarrow \mathscr{D}_{\sigma_e}^N$ by using the seed $\mathrm{KDF}(\textit{seed}|Z)$.
5: $\quad$ **for** $i = 1$ **to** ℓ **do**
6: $\quad\quad$ $\mathbf{C}_i \leftarrow \mathbf{A}_i \cdot \mathbf{e} + \mathbf{e}_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1|\ldots|\mathsf{ID}_i)$.
7: $\quad$ **end for**
8: $\quad$ $\mathbf{C}_h \leftarrow \mathbf{h} \cdot \mathbf{e} + \mathbf{e}_h$.
9: $\quad$ $\mathbf{m} \leftarrow \mathrm{Encode}(\textit{seed})$.
10: $\quad$ $\mathbf{C}_b \leftarrow \mathbf{b} \cdot \mathbf{e} + \mathbf{e}_b + \mathbf{m}$.
11: $\quad$ **return** $Z \in \{0,1\}^{256}, \mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathfrak{R}_q$.
12: **end function**

Algorithm 3.12 Latte Decrypt algorithm (at level ℓ) [ETS19].

Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \mathrm{KDF}, \mathsf{ID}_\ell, Z, (\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b), (\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1})$.
Output: μ'.

function Decrypt
  $\mathbf{V} \leftarrow \mathbf{C}_b - \mathbf{C}_h \cdot \mathbf{t}_1 - \mathbf{C}_1 \cdot \mathbf{t}_2 - \cdots - \mathbf{C}_\ell \cdot \mathbf{t}_{\ell+1}$.
  $seed' \leftarrow \mathrm{Decode}(\mathbf{V})$.
  $\mathbf{e}', \mathbf{e}'_1, \dots, \mathbf{e}'_\ell, \mathbf{e}'_h, \mathbf{e}'_b \hookleftarrow \mathscr{D}_{\sigma_e}^N$ by using the seed $\mathrm{KDF}(seed'|Z)$.
  for $i = 1$ **to** ℓ **do**
    $\mathbf{C}'_i \leftarrow \mathbf{A}_i \cdot \mathbf{e}' + \mathbf{e}'_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1|\dots|\mathsf{ID}_i)$.
  end for
  $\mathbf{C}'_h \leftarrow \mathbf{h} \cdot \mathbf{e}' + \mathbf{e}'_h$.
  $\mathbf{m}' \leftarrow \mathrm{Encode}(seed')$.
  $\mathbf{C}'_b \leftarrow \mathbf{b} \cdot \mathbf{e}' + \mathbf{e}'_b + \mathbf{m}'$.
  Check $(\mathbf{C}'_1, \dots, \mathbf{C}'_\ell, \mathbf{C}'_h, \mathbf{C}'_b)$ agrees with $(\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b)$, else return $\perp$.
  return $\mu' = Z \oplus \mathrm{KDF}(seed')$.
end function

Chapter 4

Zero-centered Discrete Gaussian Sampler

FAst, Compact, and Constant-Time (FACCT)

This chapter was published as:

- Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. FACCT: FAst, Compact, and Constant-Time Discrete Gaussian Sampler over Integers. (2019). IEEE Transactions on Computers. DOI 10.1109/TC.2019.2940949.

In this chapter, we introduce several new constant-time implementation techniques to address the efficiency issues of the binary sampling method discussed in Section 3.1.3.1. In particular, we make the following contributions:

- Our main contribution is to show that instead of storing many pre-computed $\exp(x)$ evaluations [BAA+17] or combining many Bernoulli samples [PBY17, EFGT17], the $\exp(x)$ polynomial approximation techniques with a carefully chosen precision can achieve faster and more compact constant-time implementations of the binary sampling expander. To minimise the required polynomial approximation precision, we show how to apply the Rényi divergence analysis to the binary sampling algorithm. Previous works on the Rényi divergence used a different order [BLL+15], only applied this technique to the rejection in the BLISS signing algorithm [Pre17], or applied to a different sampling method [MR18]. As opposed to [DG14], where the authors discussed the simple polynomial approximation to the $\exp(x)$ function but discarded it as inefficient in discrete Gaussian sampling, we show that with carefully chosen polynomial approximation parameters, our constant-time implementation techniques can actually be more efficient than other methods.

- We show that our scheme enjoys the property that the implementation efficiency is independent of the standard deviation. In addition, we show that our implementation techniques are flexible to integrate with existing cryptosystems, such as qTesla [ABB+19] and Falcon [PFH+17].

- As an additional independent contribution, we show how to adapt the Rényi divergence analysis to the convolution sampling algorithm and achieve smaller σ_0 for the base sampler, compared to the existing Kullback-Leibler Divergence (KLD) based algorithms [PDG14, KHR+18].

4.1 Directly Approximating the Exp Function

The Bernoulli bias p in Equation 3.1 can be directly computed within double precision (53 bits), if the RD-based relative error bound (Theorem 6) is adapted [Pre17]. The NIST PQC 1st round implementation of the lattice-based digital signature scheme Falcon [PFH+17] applied this approach to compute the rejection bias when sampling from the arbitrary-centered discrete Gaussian distribution, by using a rational function approximation of $\exp(x)$, similar to the implementation in the C standard library (see Algorithm 4.1). However, the floating-point division instructions on the Intel CPUs have various latency and throughput [Int19]. Furthermore, the compiler may replace the division operation with its own run-time library routine, which may not be constant-time [OSHG19]. Therefore, the division arithmetic should be generally avoided in constant-time implementation.

Algorithm 4.1 Rational function approximation algorithm of $\exp(x)$ [PFH+17].

Input: $x \in \mathbb{R}$, such that $|x| \leq \ln 2$.
Output: e^x with about 50-bit precision.

1: **function** $\exp(x)$
2: Let $p_1 = 1.66666666666666019037 \cdot 10^{-1}$.
3: Let $p_2 = -2.77777777770155933842 \cdot 10^{-3}$.
4: Let $p_3 = 6.61375632143793436117 \cdot 10^{-5}$.
5: Let $p_4 = -1.65339022054652515390 \cdot 10^{-6}$.
6: Let $p_5 = 4.13813679705723846039 \cdot 10^{-8}$.
7: Let $s = x/2$.
8: Let $t = s^2$.
9: Let $c = s - t \cdot (p_1 + t \cdot (p_2 + t \cdot (p_3 + t \cdot (p_4 + t \cdot p_5))))$.
10: Let $r = 1 - ((s \cdot c)/(c - 2) - s)$.
11: **return** r^2.
12: **end function**

Another classical method to compute the $\exp(x)$ is the Padé approximation [Pre17], which uses $P(x)/Q(x)$ to approximate $\exp(x)$ for some polynomials $P(x) = Q(-x)$. For the BLISS signature scheme, to satisfy the relative error bound by Theorem 6, $P(x)$ and

$Q(x)$ need to be at least degree 7 by our experiments in the `sagemath` tool. To compare $u < P(x)/Q(x)$ for some $u \hookleftarrow \mathcal{U}([0,1))$ in the rejection step of the binary sampling scheme, one may instead perform the comparison of $u \cdot Q(x) < P(x)$ to avoid the floating-point division. However, it is unclear how to choose the precision of u and the $u \cdot Q(x)$ multiplication operation for this method. Another issue is how to efficiently implement this approach in constant-time, since the implementation may involve either a high precision floating-point multiplication for $u \cdot Q(x)$ or computing the multiplication between the mantissas of u and $Q(x)$ with integer arithmetic larger than 64 bits.

We compute the $\exp(x)$ by evaluating a polynomial at point x instead, where only the floating-point additions and multiplications are involved. Both the addition and the multiplication instructions on the Intel CPUs have constant latency and throughput [Int19]. To find such an $\exp(x)$ approximation with sufficient precision, we use the following approach:

1. Let $t = y(y + 2kx)$. First, we observe that since $\sigma_0 = \sqrt{1/(2\ln 2)}$ and $\sigma = k\sigma_0$, the Bernoulli bias p in Equation 3.1 can be re-written as:

$$p = \exp(-t/(2\sigma^2)) = \exp(-\ln 2 \cdot t/k^2) = 2^{-t/k^2}.$$

   Therefore, we can find a polynomial approximation of $2^{-t/k^2}$ for $t \geq 0$.

2. Second, we adapt the method from [MBdD+10]. Let $a = -t/k^2$. We get:

$$2^a = 2^{\lfloor a \rfloor + z} = 2^{\lfloor a \rfloor} \cdot 2^z,$$

   for $0 \leq z < 1$, where z is the remaining part of rounding operation. We can directly get the multiplication with $2^{\lfloor a \rfloor}$ by changing the exponent of a floating-point variable. To approximate 2^z, we use the `sollya` tool [CJL10] to find a polynomial with sufficient number of terms, such that the minimax error is within the RD-based relative error bound.

According to the manual of the `sollya` tool [CLJ18], we use the following three functions to get such a polynomial and verify its precision:

- The guessdegree(f, I, δ, ω) function finds the minimal degree sufficient for the polynomial approximation P of function f over the interval I, such that $\|P\omega - f\|_\infty < \delta$. For example, we use the command `guessdegree(1,[0,1],1b-45,1/2^x)` to estimates the minimal degree of polynomial approximation $P(x)$ over the interval $[0,1]$, such that:

$$\|P/2^x - 1\|_\infty < 2^{-45} \implies \Delta(P\|2^x) < 2^{-45}.$$

- The fpminimax(f, n, L, I, floating, relative) function performs the heuristic from [BC07] to find a degree-n polynomial approximation P of function f over the interval I, such that P has the minimal minimax relative error, with the i-th floating-point coefficient c_i having precision L_i for all $i \leq n$. For example, we use the command `fpminimax(2^x,9,[|1,D...|],[0,1],floating,relative)` to find the polynomial approximation $P(x)$ of 2^x over the interval $[0, 1]$, with degree 9 (the result from the previous `guessdegree` command) and double precision coefficients ("`D`" represents double precision in this command). To make sure $P(0) = 1$, we set $L_0 = 1$ (1-bit precision), which results in coefficient $c_0 = 1$.

- The supnorm(p, f, I, relative, *accuracy*) function computes the interval bound $r = [\ell, u]$ for the supremum norm of the relative error $\Delta = |p/f - 1|$ over the interval I, such that $\sup_{x \in I}\{\Delta(x)\} \subseteq r$ and $0 \leq |u/\ell - 1| \leq accuracy$. For example, we use the command `supnorm(P,2^x,[0,1],relative,1b-128)` to verify $\Delta(P \| 2^x)$ over the interval $[0, 1]$ is smaller than the required relative error bound, where P is the polynomial approximation computed in the previous `fpminimax` command.

4.2 FACCT Algorithm

Our constant-time Bernoulli sampler adapting the $\exp(x)$ approximation approach above is shown in Algorithm 4.2. Let the standard deviation $\sigma = k\sigma_0$, where $k \in \mathbb{Z}^+$ and $\sigma_0 = \sqrt{1/(2 \ln 2)}$. Let $P(z)$ be the polynomial approximation of 2^z with δ_P-bit precision for $0 \leq z < 1$. Given an integer $t = y(y + 2kx)$, where $x \hookleftarrow \mathcal{D}^+_{\sigma_0}$ with tail-cut bound B and $y \hookleftarrow \mathcal{U}(\{0, \ldots, k-1\})$, this algorithm generates a sample from $\mathcal{B}_p$, where $p = \exp(-t/(2\sigma^2)) = 2^{-t/k^2}$. We assume an IEEE-754 floating-point value $f \in (0, 1]$ with $(\delta_f + 1)$-bit precision is represented by $f = (1 + mantissa \cdot 2^{-\delta_f}) \cdot 2^{exponent}$, where integer *mantissa* has δ_f bits and $exponent \in \mathbb{Z}^-$.

4.2.1 FACCT Relative Error Analysis

Here, we analyse the relative error of Algorithm 4.2. Since the algorithm will output 1 when $f = 1.0$, we only consider the case when $f \in (0, 1)$, which implies $exponent < 0$. Let $\mathcal{P}_{\text{FACCT}}$ and $\mathcal{P}_{\text{IDEAL}}$ represent the distribution of the FACCT Bernoulli sampler and the ideal Bernoulli sampler, respectively. Since $a = -t/k^2$ and $z = a - \lfloor a \rfloor$, we have:

$$\mathcal{P}_{\text{IDEAL}}(\lfloor a \rfloor, z) = \exp(-t/(2\sigma^2)) = 2^a = 2^{z + \lfloor a \rfloor}.$$

Algorithm 4.2 FACCT Bernoulli sampler.

Input: Deviation $\sigma = k\sigma_0$, where $k \in \mathbb{Z}^+$ and $\sigma_0 = \sqrt{1/(2\ln 2)}$. Integer $t = y(y + 2kx)$, where $x \hookleftarrow \mathscr{D}^+_{\sigma_0}$ with tail-cut bound B and $y \hookleftarrow \mathscr{U}(\{0, \ldots, k-1\})$. Polynomial approximation $P(z)$ of 2^z with δ_P-bit precision for $0 \le z < 1$. Bit length $\ell \ge 2B + 1$.

Output: A sample from $\mathscr{B}_p$, where $p = \exp(-t/(2\sigma^2)) = 2^{-t/k^2}$.

1: **function** BernoulliSampler(t)
2: Let $a = -t/k^2$.
3: Let $z = a - \lfloor a \rfloor$.
4: Evaluate $s = P(z)$ on point z.
5: Let $f = s \cdot 2^{\lfloor a \rfloor}$.
6: Let represent f by $f = (1 + mantissa \cdot 2^{-\delta_f}) \cdot 2^{exponent}$, with δ_f-bit *mantissa*.
7: Sample $r_m \hookleftarrow \mathscr{U}(\{0,1\}^{\delta_f+1})$.
8: Sample $r_e \hookleftarrow \mathscr{U}(\{0,1\}^{\ell})$.
9: **if** ($r_m < mantissa + 2^{\delta_f}$ **and** $r_e < 2^{\ell+exponent+1}$) **or** $f = 1.0$ **then**
10: **return** 1.
11: **else**
12: **return** 0.
13: **end if**
14: **end function**

Theorem 12 (Adapted from [MBdD+10], Def. 5.1 and Eq. 5.7)**.** *The absolute error between an accurate polynomial evaluation $P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ and the evaluation $H(x)$ by using Horner's rule with δ-bit precision floating-point arithmetic is:*

$$|H(x) - P(x)| \le \gamma_{2n} \cdot \sum_{i=0}^{n} |a_i| \cdot |x|^i,$$

where $\gamma_{2n} \approx 2n \cdot 2^{-\delta}$ if $2n$ is much smaller than $1/2^{-\delta}$.

The polynomial approximation P from the `sollya` tool is evaluated by using Horner's rule (see Figure 4.1 for an example). Assume a degree-n polynomial approximation P only has positive coefficients. Since we use (δ_f+1)-bit precision floating-point arithmetic to compute $P(z)$ for $0 \le z < 1$, by adapting Theorem 12, we have:

$$\begin{aligned}
\mathscr{P}_{\text{FACCT}}(\lfloor a \rfloor, z) &= \frac{(mantissa + 2^{\delta_f})}{2^{\delta_f+1}} \cdot \frac{2^{\ell+exponent+1}}{2^{\ell}} \\
&= (1 + mantissa \cdot 2^{-\delta_f}) \cdot 2^{exponent} \\
&= f \\
&\le (1 + 2n \cdot 2^{-(\delta_f+1)})(P(z) \cdot 2^{\lfloor a \rfloor}),
\end{aligned}$$

where the last inequality follows because $|H(z)/P(z) - 1| \le \gamma_{2n}$ and $\gamma_{2n} \approx 2n \cdot 2^{-(\delta_f+1)}$ due to Theorem 12 (The multiplication with $2^{\lfloor a \rfloor}$ does not change the error since we directly change the exponent of a floating-point variable). Then, the relative error Δ between

$\mathscr{P}_{\text{FACCT}}$ and $\mathscr{P}_{\text{IDEAL}}$ is:

$$\begin{aligned}
\Delta &= \max_{\lfloor a \rfloor, z} \left| \frac{\mathscr{P}_{\text{FACCT}}(\lfloor a \rfloor, z)}{\mathscr{P}_{\text{IDEAL}}(\lfloor a \rfloor, z)} - 1 \right| \\
&\leq \max_{\lfloor a \rfloor, z} \left| \frac{(1 + 2n \cdot 2^{-(\delta_f+1)})(P(z) \cdot 2^{\lfloor a \rfloor})}{2^{z+\lfloor a \rfloor}} - 1 \right| \\
&\leq (1 + n \cdot 2^{-\delta_f})(1 + 2^{-\delta_P}) - 1 \quad \text{(by definition of } \delta_P\text{)} \\
&= 2^{-\delta_P} + n \cdot (2^{-\delta_f} + 2^{-(\delta_P+\delta_f)}). \qquad (4.1)
\end{aligned}$$

We also need to make sure that $\ell + \mathit{exponent} + 1 \geq 0$ during the comparison in Algorithm 4.2. Let Δ be the relative error in Equation 4.1. Since $a = -t/k^2$, by definitions of *exponent* and Δ from Equation 4.1, we have:

$$\begin{aligned}
\mathit{exponent} &\geq \lfloor \log_2((1 - \Delta) \cdot 2^{-t/k^2}) \rfloor \\
&\geq \lfloor -1 - t/k^2 \rfloor \quad \text{(we make } \Delta \leq 1/2\text{)} \\
&\geq \left\lfloor -1 - \frac{y(y + 2kx)}{k^2} \right\rfloor \quad \text{(by definition of } t\text{)} \\
&\geq \left\lfloor -1 - \frac{y^2}{k^2} - \frac{2kxy}{k^2} \right\rfloor \\
&\geq -2B - 2 \quad \text{(by definitions of } x \text{ and } y\text{)}.
\end{aligned}$$

Therefore, if $\ell + \mathit{exponent} + 1 \geq 0$, we have:

$$\ell \geq 2B + 1. \qquad (4.2)$$

To ensure that the compiler will not replace any floating-point arithmetic with its own run-time library implementation, we manually write the arithmetic in the source code by using constant-time instructions with the Intel intrinsics. This also enables the SIMD instruction sets, such as the AVX2, which computes 4x double precision floating-point arithmetic in parallel.

Compared with the previous table-based constant-time Bernoulli sampling techniques [BHLY16, PBY17, EFGT17, BAA+17], where the number of table entries is proportional to the bit length of t, our implementation is more compact in terms of the memory consumption, since we only need to store a small number of polynomial coefficients. For example, in the BLISS-I parameter set, $k = 254$, which implies $0 \leq t < 2^{21}$. This requires at least 21 table entries in the previous techniques, compared to only 9 coefficients for about 45-bit precision in our implementation (see Section 4.4). Also, our implementation is more efficient for large standard deviations, since the code is independent of σ

(assuming $-1/k^2$ is a pre-computed constant), while the number of iterations (proportional to the number of table entries) relies on k in previous table-based approaches. In addition, if the application requires samples from several different standard deviations, our implementation does not need additional pre-computed tables for each different k.

4.2.2 AVX2 Implementation

In order to implement the binary sampling scheme (Algorithm 3.1) with our FACCT Bernoulli sampler (Algorithm 4.2) by using the AVX2 instructions, since each sample is independent i.e. data and arithmetic operations are independent between samples, we can simply vectorise the arithmetic operations in Algorithm 3.1 and compute the rejection conditions by using the vectorised Algorithm 4.2 in batch. We then discard the rejected samples and continue the batch sampling process until we generate the required amount of samples. Since the sampler needs to generate and apply a sign bit for each sample, we choose 8 as the batch size, in which case an extra random byte is generated during each batch and the 8 bits of this random byte become the sign bits of 8 samples in the batch. Each arithmetic operation performed in the batch is parallelised by two 4×64-bit AVX2 instructions. Note that since our main purpose of using the AVX2 instructions is to avoid the non-constant time floating-point code generated by the compiler, maximising the performance is not the priority. Therefore, our AVX2 implementation may not be optimal in terms of the speed since our implementation does not consider the latency and throughput of the AVX2 instructions [Sei18, CHK+21] or the pipelines of the target CPU [BHK+22].

4.3 Concrete Rényi Divergence Based Convolution Sampling

Prest only implied the potentially tighter parameters for the convolution theorem based samplers by adapting the Rényi divergence [Pre17]. In this section, we discuss the concrete parameter choice for the RD-based convolution sampling scheme.

Recall that $ML(\mathcal{P}\|\mathcal{Q}) \approx \Delta(\mathcal{P}\|\mathcal{Q})$ when $\Delta(\mathcal{P}\|\mathcal{Q}) \to 0$ (Lemma 4.2 in [MW17]). Also, in the convolution sampler adapting Theorem 10, typically $\mathbf{z} = (k-1, k)$ for some $k \geq 4$ [MW17]. Therefore, by applying Theorem 6, we provide the following concrete RD-based parameter choice lemmas:

Lemma 1. *Let $x_1, x_2 \hookleftarrow \mathcal{D}'_{\sigma_0}$, with $\sigma_0 = \sigma/\sqrt{(k-1)^2+k^2}$ for some $\sigma \in \mathbb{R}^+$ and $k \geq 4$. If $\sigma_0 \geq k\eta_\epsilon(\mathbb{Z})/\sqrt{\pi}$ and $\Delta(\mathcal{D}'_{\sigma_0}\|\mathcal{D}_{\sigma_0}) \leq \mu$, then for M independent samples, sampling from the distribution $\mathcal{P}$ of $(k-1)x_1 + kx_2$ will be λ-bit secure, if $\Delta(\mathcal{P}\|\mathcal{D}_\sigma) \leq 2\epsilon + 2\mu \leq \sqrt{1/(4\lambda \cdot M)}$.*

Lemma 2. *Let $x_1, x_2 \hookleftarrow \mathcal{D}'_{\sigma_0}$, with $\sigma_0 = \sigma/\sqrt{1+k^2}$ for some $\sigma \in \mathbb{R}^+$ and $k \geq 2$. If:*

$$\sigma_0 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}, \quad \sqrt{\frac{1}{\sigma_0^{-2} + (k\sigma_0)^{-2}}} \geq k\eta_\epsilon(\mathbb{Z})/\sqrt{2\pi},$$

and $\Delta(\mathcal{D}'_{\sigma_0} \| \mathcal{D}_{\sigma_0}) \leq \mu$, then for M independent samples, sampling from the distribution $\mathcal{P}$ of $x_1 + kx_2$ will be λ-bit secure, if $\Delta(\mathcal{P}\|\mathcal{D}_\sigma) \leq 4\epsilon + 2\mu \leq \sqrt{1/(4\lambda \cdot M)}$.

Proof. We show that for distributions $\mathcal{P}$ and $\mathcal{Q}$, and M independent samples, sampling from $\mathcal{P}$ will be λ-bit secure, if $ML(\mathcal{P}\|\mathcal{Q}) \leq \sqrt{1/(4\lambda \cdot M)}$. Let $\alpha = 2\lambda$. By combining Theorem 6 and Theorem 7, we get:

$$R_{2\lambda}(\mathcal{P}\|\mathcal{Q}) \leq 1 + \lambda \cdot (ML(\mathcal{P}\|\mathcal{Q}))^2 \leq 1 + 1/(4M) \implies ML(\mathcal{P}\|\mathcal{Q}) \leq \sqrt{1/(4\lambda \cdot M)}.$$

Then, let $\sigma_0 = \sigma/\sqrt{(k-1)^2 + k^2}$, $\mathbf{z} = (k-1, k)$, and $\sigma = (\sigma_0, \sigma_0)$ in Theorem 10 to get $\Delta(\mathcal{P}\|\mathcal{D}_\sigma) \leq 2\epsilon + 2\mu$. Let $\sigma_0 = \sigma/\sqrt{1+k^2}$, $\sigma_1 = \sigma_0$, and $\sigma_2 = k\sigma_0$ in Theorem 11, we get $\Delta(\mathcal{P}\|\mathcal{D}_\sigma) \leq 4\epsilon + 2\mu$. We replace ML with Δ in both Theorem 10 and Theorem 11, then get Lemma 1 and Lemma 2, respectively. □

Since the constraint for σ_0 in Lemma 2 is looser than in Lemma 1 (about $\sqrt{2}$ times), but σ_0 shrinks faster in Lemma 1 instead, one can apply both lemmas on different recursion levels. For example, one may adapt Lemma 1 on all the intermediate levels and use Lemma 2 on the bottom level, to achieve possibly smaller base sampler deviation.

The total relative errors for different number of convolution levels are shown in Table 4.1. Typically, the standard deviations in lattice-based cryptosystems require 3 levels at maximum. Let Δ be the relative error of the base sampler. The "Mixed-k" in Table 4.1 represents the example we discussed above.

For $\sigma \approx 215$ in the BLISS-I parameter set, the base sampler deviation σ_0 and convolution parameters $\mathbf{z}_i$ are shown in Table 4.2 for $i \leq \ell$, where ℓ is the number of convolution levels. We assume $M = m \cdot q_s$ with $m = 1024$, $q_s = 2^{64}$, $\lambda = 128$, and $\Delta \leq 2^{-53}$. Compared with the KLD-based convolution schemes [PDG14, KHR+18, KRR+18], our RD-based convolution parameter choice lemmas generate smaller base sampler deviations for the same number of convolution levels.

Table 4.1: Total Relative Errors for Different Number of Convolution Levels.

$\mathbf{z}$	1 Level	2 Levels	3 Levels
$(k-1, k)$	$2\epsilon + 2\Delta$	$6\epsilon + 4\Delta$	$14\epsilon + 8\Delta$
$(1, k)$	$4\epsilon + 2\Delta$	$12\epsilon + 4\Delta$	$28\epsilon + 8\Delta$
Mixed-k	$4\epsilon + 2\Delta$	$10\epsilon + 4\Delta$	$22\epsilon + 8\Delta$

Table 4.2: Convolution Parameters for $\sigma \approx 215$.

Method	ℓ	σ_0	$\mathbf{z}_i$
KLD	1	19.53	$\mathbf{z}_1 = (1, 11)$
KLD	2	6.18	$\mathbf{z}_1 = (1, 11), \mathbf{z}_2 = (1, 3)$
RD $(k-1, k)$	1	17.92	$\mathbf{z}_1 = (8, 9)$
RD $(1, k)$	2	5.67	$\mathbf{z}_1 = (1, 12), \mathbf{z}_2 = (1, 3)$
RD Mixed-k	2	5.67	$\mathbf{z}_1 = (8, 9), \mathbf{z}_2 = (1, 3)$

4.4 Evaluation

We implement the binary sampling scheme (Algorithm 3.1) by combining the constant-time CDT base sampler with the FACCT Bernoulli sampler (Algorithm 4.2). We choose the tail-cut bound B by Theorem 5, which guarantees that the R_∞ between the tail-cut and the ideal discrete Gaussian is $\leq \exp(1)$ over all $M = m \cdot q_s$ samples, corresponding to a loss of at most $\log_2(\exp(1)) \approx 1.44$ bits of security for the tail-cut samples relative to the ideal discrete Gaussian sampling case. On the other hand, we choose $\Delta_{\mathscr{D}^+_{\sigma_0}}$ and $\Delta_{\mathscr{B}_p}$ by Theorem 6, which guarantees that we lose at most 1 bit of security due to the relative precision errors, respectively. Hence overall our choice of tail-cut and precision parameters ensure that we lose at most $1 + 1 + 1.44 = 3.44$ bits of security with respect to the ideal discrete Gaussian sampling over M samples. Since our FACCT Bernoulli sampler is independent of σ, we pick the precision δ_P of the polynomial approximation and bit length ℓ in Algorithm 4.2 by using Equation 4.1 and Equation 4.2, respectively. We use double precision floating-point, where the mantissa has $\delta_f = 52$ bits, and we fix the parameters in Table 4.3 for our implementations in the benchmarks.

Table 4.3: Parameters for Implementations.

Parameter	Description	Value
M	Number of discrete Gaussian samples	2^{74}
λ	Security level	128
B	Tail-cut bound	9
δ_P	Precision of the polynomial approximation	45
δ_f	Number of bits in the mantissa	52
ℓ	Bit length in Algorithm 4.2	19
$\Delta_{\mathscr{D}^+_{\sigma_0}}$	Relative error of the base sampler	2^{-46}
$\Delta_{\mathscr{B}_p}$	Relative error of the Bernoulli sampler	$2^{-44.99}$

We employ the full-table access CDT base sampler. We select the parameters in Table 4.3 such that the base sampler has about 126-bit absolute precision. We store each CDT entry in two 63-bit integers, then the constant-time comparison of $x < y$, where $0 \leq x, y < 2^{63}$, can be performed by a 64-bit signed integer subtraction, since the sign bit of $x - y$ will be 1 when $x < y$. We compute the CDT in reversed order such that $\mathscr{P}(i) = \mathrm{CDT}[i] - \mathrm{CDT}[i+1]$ for $i \in [0, B]$, where the subtraction only enlarges the relative error by a factor of about

σ_0 [PDG14, MR18]. For the uniform sampling over the range $[0, k-1]$, we adapt similar techniques as in [SSZ18] to reduce the rejection rate. We generate random integers over a larger range $[0, 2^\ell - 1]$ instead, where $2^\ell > k$, and then perform the modulo k operations. In addition, we show how to get the polynomial approximation P in our FACCT sampler implementation by using the `sollya` tool in Figure 4.1, and we verify $\Delta(P||2^x) < 2^{-45.9}$ over the interval $[0, 1]$.

Figure 4.1: The polynomial approximation P in the FACCT sampler implementation.

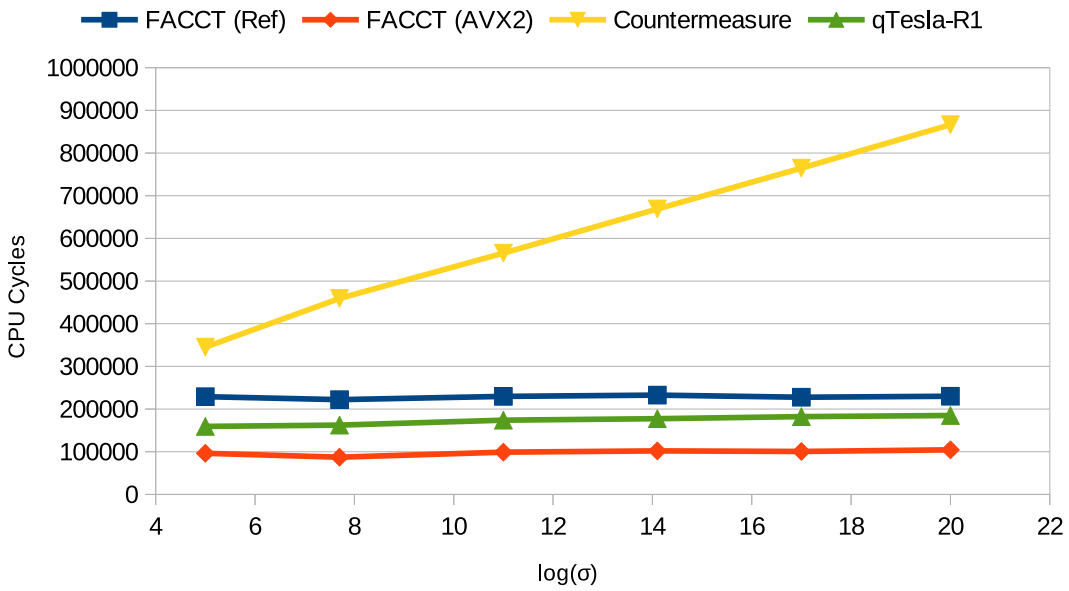
```
> guessdegree(1,[0,1],1b-45,1/2^x);
[9;9]
> P=fpminimax(2^x,9,[|1,D...|],[0,1],floating,relative);
> P;
1 + x * (0.693147180561933806686170100874733179807662963867 19
+ x * (0.240226506876527745593108420507633127272129058837 89
+ x * (5.550410984131824709830738129312521778047084808349 6e-2
+ x * (9.618120933175645231871797591338690835982561111450 2e-3
+ x * (1.333387755250109744584174897852335561765357851982 1e-3
+ x * (1.539604321053863805399131159390435641398653388023 4e-4
+ x * (1.535991421946201169828304100573035384513786993920 8e-5
+ x * (1.230394437555541324973693885491687893818379961885 5e-6
+ x * 1.432910037894390942758726138761549151467988849617 54e-7))))))))
> supnorm(P,2^x,[0,1],relative,1b-128);
[1.491806901685506403985743728294477543016355700589 2e-14;
1.491806901685506403985743728294477543020602726225 8424e-14]
```

For the benchmarks, we select $\sigma \approx \{2^5, 215, 2^{11}, 17900, 2^{17}, 2^{20}\}$, where 215 (approximately $2^{7.7}$) and 17900 (approximately $2^{14.1}$) are the standard deviations from the BLISS-I [DDLL13] and the Dilithium-G [DLL+17] recommended parameter sets[1], respectively. We compare the run-time of our implementations with the binary sampling scheme from qTesla-R1 [BAA+17] and the countermeasures from [PBY17, EFGT17]. Since the countermeasures did not have a full implementation code available, we simply replace the Bernoulli sampling subroutine in our non-AVX2 reference implementation with the countermeasures. Because the optimal convolution sampling schemes [KRR+18, KRVV19] require major refactoring of the bitslicing base sampler for each different σ, we exclude it from this benchmark. We use the AES256 counter mode with hardware AES instructions (AES-NI) [Gue09] to generate the randomness in all the implementations. We use `clang` 8.0.0 to compile our AVX2 implementation, and

[1]The Dilithium-G parameter sets are availabe at the old version of the preprint from the authors: https://eprint.iacr.org/eprint-bin/getfile.pl?entry=2017/633&version=20170627:201152&file=633.pdf.

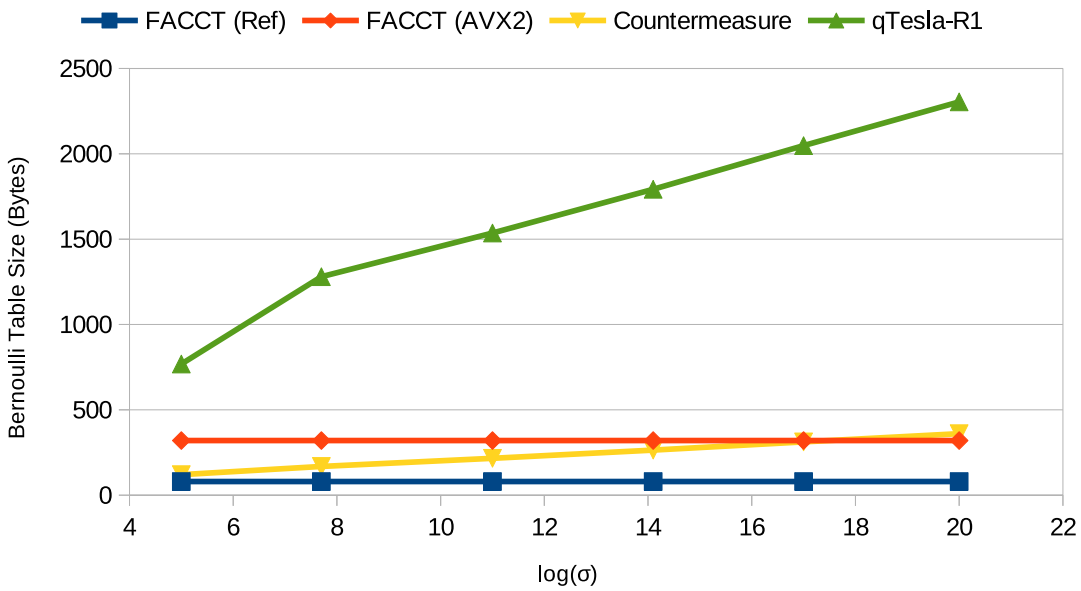
use `gcc 9.1.1` to compile all the other implementations, with the compiling options `-O3 -march=native` enabled for both compilers. The benchmark is running on an Intel i7-7700K CPU at 4.2GHz, with the Hyperthreading and the Turbo Boost disabled. We generate $m = 1024$ samples for 1000 times and measure the median number of the consumed CPU cycles. The comparison results are shown in Figure 4.2. The "Ref" in the following figures and tables represents non-AVX2 reference implementations. We implement the non-AVX2 reference implementations by using the constant-time SSE4 floating-point instructions, which is available on the Intel Nehalem architecture back to 2008 and all subsequent Intel architectures. However, older Intel CPUs such as the Pentium III may not support constant-time hardware floating-point multiplication instructions [BBE+19]. In addition, our techniques should be portable to other architectures supporting constant-time hardware floating-point instructions (addition and multiplication). However, on architectures where the floating-point unit is unavailable, the floating-point arithmetic will be emulated by the software routines from the C run-time library, which is usually not constant-time [OSHG19]. One may consider to use the variant from the subsequent work GALACTICS [BBE+19] instead on these architectures, which computes the polynomial approximation results in fixed-point arithmetic by only using integers.

Figure 4.2: Comparison of the CPU cycles for different σ.



We measure the table size of the Bernoulli sampler by computing the number of table entries times the size of the variable type (in bytes) for each implementation. Since we store vectors instead of single values in our AVX2 implementation, the table size is 4x our non-AVX2 reference implementation. The comparison results are shown in Figure 4.3.

Figure 4.3: Comparison of the Bernoulli table size for different σ.



From Figure 4.2, compared to the countermeasures, our non-AVX2 reference implementation is 1.5x–3.7x faster, and our AVX2 implementation is 3.5x–8.3x faster, respectively, especially for the larger σ. In addition, our AVX2 implementation is 1.6x–1.8x faster than the qTesla-R1 sampler. Note that our non-AVX2 reference implementation is suboptimal on the run-time speed, since the constant-time floating-point arithmetic instructions for a single value have similar latencies and throughputs as their SIMD counterparts on the Intel CPUs [Int19]. Therefore, our optimal AVX2 implementation should be used if run-time speed is concerned.

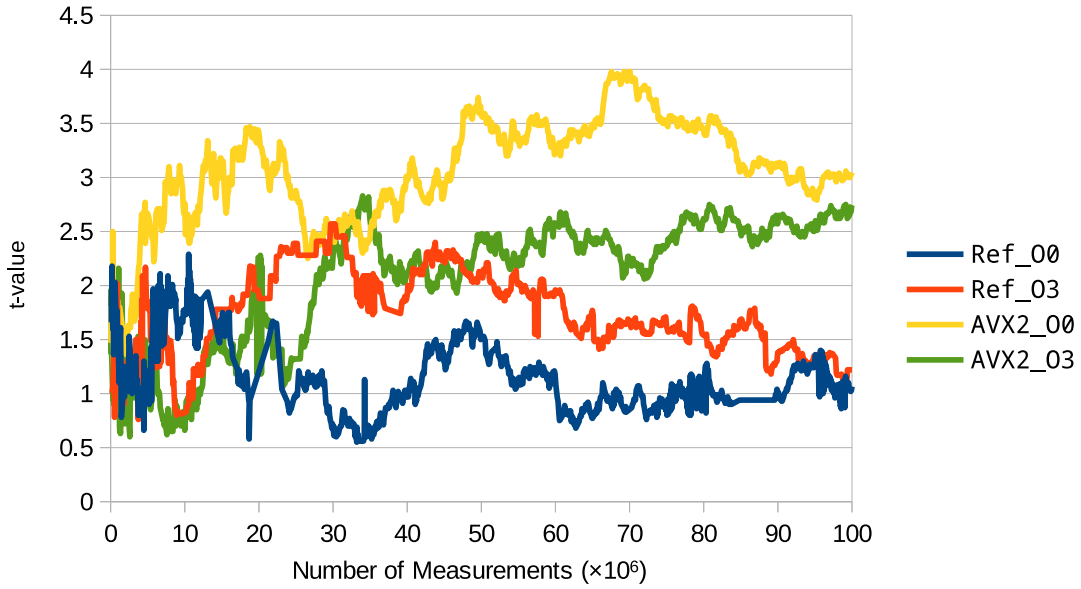
From Figure 4.3, our implementations have much smaller table sizes than the qTesla-R1 sampler (9.6x–28.8x for our non-AVX2 reference implementation and 2.4x–7.2x for our AVX2 implementation), especially for the larger σ. In addition, compared to the countermeasures, our non-AVX2 reference implementation has 1.5x–4.5x smaller table size, and our AVX2 implementation has similar table size, respectively.

From both Figure 4.2 and Figure 4.3, we also verify that the efficiency of our implementations is independent of σ.

We also verify that our implementations of the FACCT Bernoulli sampler in Algorithm 4.2 are constant-time by using the `dudect` tool [RBV17]. The `dudect` tool first takes a number of execution timing measurements for a program by using two different sets of input data: a fixed class with constant values e.g. zeros as the input and a random class with fresh random input data during each measurement. Then, the `dudect` tool checks whether the distributions between the two sets of measurements are statistically different, by performing the Welch's t-test against the null hypothesis

that the distributions of timing measurements between the fixed and the random classes are the same. The non-constant time threshold for the t-value is 10 in the `dudect` tool i.e. if the t-value is higher than 10, then the program is categorised as non-constant time. Since the compiler optimisations may accidentally hide[2] the timing leakage, we test our implementations compiled with both `-O3 -march=native` (full optimisations, same as in our benchmarks) and `-O0 -march=native` (no compiler optimisations) options. We compile the non-AVX2 reference implementation by using `gcc`, and compile the AVX2 implementation by using `clang`. We take up to 100 million measurements, and the results are demonstrated in Figure 4.4. From Figure 4.4, for both compiler optimisation options `-O0` and `-O3`, the t-values for our non-AVX2 reference implementation and the AVX2 implementation of the FACCT Bernoulli sampler are less than 4 for up to 100 million measurements, which is well below the non-constant time threshold. On the contrary, the t-value of the non-constant time Bernoulli sampler (Algorithm 3.3) from the original BLISS implementation[3] [DDLL13] compiled with the `-O0` option quickly exceeds the non-constant time threshold within 20000 measurements in our experiment.

Figure 4.4: t-values from the timing measurements of the FACCT Bernoulli sampler.



[2] See the "Further notes" at https://github.com/oreparaz/dudect/blob/master/README.md.
[3] https://bliss.di.ens.fr/bliss-06-13-2013.zip.

4.5 Applications

In this section, we compare the performance of our software implementations[4] with previous implementations from actual cryptosystems.

4.5.1 Sampling from the BLISS-I Standand Deviation

In this section, we compare the performance of our FACCT sampler implementations with the sampler implementations from qTesla [BAA+17, ABB+19], the bitslicing convolution scheme [KRR+18], and previous countermeasures [PBY17, EFGT17], using the BLISS-I parameter set. Since other convolution schemes [PDG14, KHR+18] only have hardware implementations, we only compare with the software implementation of the bitslicing convolution [KRR+18].

The BLISS-I parameter set has $k = 254$ and $\sigma \approx 215$. We use the similar benchmark setup as Section 4.4, with $\lambda = 128$, $m = 1024$, and $q_s = 2^{64}$, which gives the same number of samples M as in Table 4.3. For the bitslicing convolution scheme, we compare with the implementation of 128-bit absolute precision. We directly use the benchmark script[5] from the authors to measure the number of the CPU cycles of generating 64 base samples, and scale the result up to $4m = 4096$ base samples. We also scale this number by the same factor as in [KRR+18] to retrieve the AVX2 result. The CPU cycles are shown in Table 4.4.

Table 4.4: Comparison of the CPU Cycles for Generating $m = 1024$ Samples from $\mathcal{D}_{\mathbb{Z},\sigma}$, with $\sigma \approx 215$.

Scheme	CPU Cycles (Ref)	CPU Cycles (AVX2)
qTesla-R1	162215	–
qTesla-R2	2531610	–
Bitslicing	$\approx$ 532800	$\approx$ 254708
Countermeasure	459297	–
FACCT	221991	87192

To measure the memory consumption of each implementation, we compute the table sizes for both the base samplers and the Bernoulli samplers by using similar approaches as in Section 4.4. Since the bitslicing approach does not require a table, but has a rather large code size [KRR+18], for a fair comparison, we also measure the assembly code size (in bytes) of the sampling functions. We compile the source codes by using the compiling options `-Os -march=native` to generate more compact assembly code, and use the `objdump` command to perform the disassembly. The memory consumption comparison

[4] The implementation source codes are available at https://gitlab.com/raykzhao/gaussian.
[5] https://github.com/Angshumank/const_gauss.

results are shown in Table 4.5. The "Table" represents the total table size, and for binary sampling variants, the results are in the form of "base sampler table size+Bernoulli sampler table size". The "Code" represents the code size, and the "Total" represents the sum of the table size and the code size.

Table 4.5: Comparison of the Memory Consumption for $\sigma \approx 215$.

Scheme	Table (Bytes)	Code (Bytes)	Total (Bytes)
qTesla-R1	192+1280	597	2069
qTesla-R2	90816	961	91777
Bitslicing	–	$\approx$ 98816	$\approx$ 98816
Countermeasure	144+168	440	752
FACCT (Ref)	144+80	659	883
FACCT (AVX2)	576+320	1275	2171

From Table 4.4, in addition to the results from Figure 4.2, our implementations significantly outperform the bitslicing convolution scheme (2.4x for the reference implementation and 2.9x for the AVX2 implementation). Our implementations are also significantly faster than the qTesla-R2 sampling algorithm for larger $\sigma \approx 215$ (11.4x for the reference implementation and 29.0x for the AVX2 implementation).

From Table 4.5, in addition to the results from Figure 4.3, our non-AVX2 reference implementation consumes 2.3x smaller memory space than the qTesla-R1 sampler, and has similar memory consumption compared to the countermeasures, respectively. Our AVX2 implementation has similar memory consumption compared to the qTesla-R1 sampler. However, for larger σ, as shown in Figure 4.3, the qTesla-R1 sampler will consume significantly more memory space to store the Bernoulli table, while our implementations maintain similar memory consumption. Both of our implementations consume much smaller memory space than the bitslicing convolution scheme (111.9x for the non-AVX2 reference implementation and 45.5x for the AVX2 implementation). In addition, our implementations have much lower memory consumption compared to the qTesla-R2 sampler for larger $\sigma \approx 215$ (103.9x for the non-AVX2 reference implementation and 42.2x for the AVX2 implementation).

4.5.2 qTesla

To test the run-time speed of our sampler in a cryptosystem, we replace the sampler in the AVX2 implementation of qTesla-R2 with our FACCT AVX2. Since the cSHAKE [NIS16b] software random generator used by qTesla-R2 is much slower than the AES-NI [Gue09], we measure the performance after replacing the random generator of the sampler with the AES-NI in the implementations. The CPU cycles measured by the benchmark script from qTesla-R2 are shown in Table 4.6. The qTesla-R2 KeyGen with

our AVX2 sampler (AES-NI) is 2.3x–2.8x faster than the original implementations (modified to use AES-NI instead of cSHAKE). Note that the standard deviations in qTesla-R2 ($\sigma \approx 10.2$–22.93) is smaller than the deviations in previous benchmarks. Therefore, our implementation maintains good performance even for smaller σ.

Table 4.6: Comparison of the CPU Cycles for qTesla-R2 (AVX2) KeyGen.

Scheme	Original (cSHAKE)	Original (AES-NI)	FACCT
qTesla-I	1093917	1009155	402022
qTesla-III	2875728	2419416	1039426
qTesla-V	14352751	11607570	4007417

4.5.3 Falcon

To test the performance of our proposed constant-time $\exp(x)$ implementation in Section 4.1, we replace the $\exp(x)$ in Falcon [PFH+17] with our non-AVX2 reference implementation. Since the $\exp(x)$ is used when performing the rejection sampling from the arbitrary-centered discrete Gaussian in the signing, we measure the signing speed by using the benchmark script from Falcon. The results are shown in Table 4.7. Our constant-time $\exp(x)$ reference implementation only adds very slight overhead (6.5%–8.2%) to the signing (However, the rejection rate of sampling may still be secret dependent).

Table 4.7: Signing Speed Comparison for Falcon.

N	Original (sig/s)	Our Implementation (sig/s)
256	17844.457	16375.575
384	10232.885	9561.668
512	8781.839	8076.828
768	5281.893	4933.550
1024	4443.585	4086.705

4.6 Research Impact

Recently, GALACTICS [BBE+19], a fully constant-time implementation of the BLISS signature scheme, adapted a variant of our FACCT sampling algorithm without floating-point arithmetic. The polynomial approximation in this implementation has the same degree 9 as our FACCT sampler and employs only the integer arithmetic, which is more suitable for architectures where constant-time hardware floating-point instructions (addition and multiplication) or even the floating-point unit may not be available. However, the techniques from [BBE+19] require a different polynomial approximation for each different σ and therefore lose the σ-independent feature of our FACCT sampling algorithm.

In addition, Howe et al. adapted a variant of our constant-time $\exp(x)$ implementation as a subroutine of their constant-time discrete Gaussian sampler [HPRR20]. Their constant-time discrete Gaussian sampler is employed in the latest constant-time Falcon digital signature implementation [PRR19] (a NIST PQC 3$^{\text{rd}}$ round finalist). In addition, the authors of the constant-time Falcon implementation [PRR19] have verified that the relative error of our constant-time $\exp(x)$ is less than 2^{-45}. Very recently, Mera et al. employed our FACCT sampler in the implementation of their lattice-based functional encryption scheme [MKMS21].

Chapter 5

Arbitrary-centered Discrete Gaussian Sampler

COmpact and Scalable Arbitrary-Centered (COSAC)

This chapter was published as:

- Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and Scalable Arbitrary-Centered Discrete Gaussian Sampling over Integers. (2020). Proceedings of PQCrypto 2020. DOI 10.1007/978-3-030-44223-1_16.

In this chapter, we introduce a novel arbitrary-centered discrete Gaussian sampling algorithm over integers by generalising ideas from [Dev86]. Our scheme samples from a continuous normal distribution and performs rejection sampling on rounded samples by adapting techniques from [HLS18, ZCHW17]. Compared to previous arbitrary-centered discrete Gaussian sampling techniques discussed in Chapter 3, our scheme has the following advantages:

- Our sampling algorithm does not require any pre-computations related to a specific discrete Gaussian distribution or a specific standard deviation, and both the center and the standard deviation can be arbitrary determined on-the-fly at runtime.
- In addition, we show in Section 5.1 that our sampling method only requires a low number of trials close to 2 per sample on average compared to about 8–10 on average in the rejection sampling with regards to a uniform distribution, and the rejection rate of our algorithm decreases when scaling up σ. Therefore, our sampling algorithm is not limited to small σ and can be adapted to sample from larger σ without affecting the efficiency.

- Since sampling from a continuous normal distribution is a well-studied topic [TLLV07] and the sampling algorithms are implemented in many existing software libraries (including the C++11 STL) and hardware devices, one can easily implement our scheme by employing existing tools.
- We provide a center-independent run-time implementation of our algorithm without timing leakage of the center and it can be adapted to achieve constant-time implementation of convolution-style lattice trapdoor sampler [MP12, Pei10] and IBE [BFRS18].

5.1 COSAC Algorithm

Devroye defined a variant of the discrete Gaussian distribution as $\Pr[X = z] = C \cdot \exp(-(|z| + 1/2)^2/(2\sigma^2))$ [Dev86], where $z \in \mathbb{Z}$ and C is the normalisation constant, i.e. $\Pr[X = z] \propto \rho_{-1/2,\sigma}(z)$ for $z \geq 0$ and $\Pr[X = z] \propto \rho_{1/2,\sigma}(z)$ for $z < 0$, where $\rho_{c,\sigma}(z)$ is the 1-dimensional (continuous) Gaussian function with center c and standard deviation σ as defined in Section 2.1. A rejection sampling algorithm (see Algorithm 5.1) was provided by [Dev86] with rejection probability less than $(2/\sigma)\cdot\sqrt{2/\pi}$ for such a distribution, which is fast for large σ.

Algorithm 5.1 Rejection sampler adapted from [Dev86], pg. 117, ch. 3.

Input: Standard deviation $\sigma \in \mathbb{R}^+$.
Output: A sample z distributed as $\Pr[X = z] = C \cdot \exp(-(|z| + 1/2)^2/(2\sigma^2))$.

1: **function** Sampler(σ)
2: Sample $x \hookleftarrow \mathcal{N}(0, \sigma^2)$.
3: Sample $r \hookleftarrow \mathcal{U}([0, 1))$.
4: Let $Y = (|\lfloor x \rceil| + 1/2)^2 - x^2$.
5: **if** $r < \exp(-Y/(2\sigma^2))$ **then**
6: Let $z = \lfloor x \rceil$.
7: **else**
8: **goto** 2.
9: **end if**
10: **return** z.
11: **end function**

Here, we generalise Algorithm 5.1 to sample from $\mathcal{D}_{c,\sigma}(z)$. By removing the absolute value and replacing the fixed center $-1/2$ with a generic center c in Algorithm 5.1, we observe that if $(c \geq 1/2, x \geq 0)$ or $(c \leq -1/2, x < 0)$, then $Y' = (\lfloor x \rceil + c)^2 - x^2 \geq 0$.[1] Therefore, we can replace Y with Y' and perform a similar rejection sampling to Algorithm 5.1 when sampling from $\mathcal{D}_{c,\sigma}(z)$ for some c and $z = \lfloor x \rceil$. To extend Algorithm 5.1 to support all $c \in \mathbb{R}$ and $z \in \mathbb{Z}$, we first compute $c_I = \lfloor c \rceil$ and $c_F = c_I - c$, where $c_F \in [-1/2, 1/2]$.

[1] The conditions of c, x here when $Y' \geq 0$ are not exhaustive.

Then, we can sample from $\mathscr{D}_{-c_F,\sigma}$ instead, since $\mathscr{D}_{c,\sigma} = \mathscr{D}_{-c_F,\sigma} + c_I$. To sample from $\mathscr{D}_{-c_F,\sigma}$ for all $c_F \in [-1/2, 1/2]$, we instinctively[2] shift the center of the underlying continuous normal distribution by ± 1, i.e. sampling $y \hookleftarrow \mathcal{N}(c_N, \sigma^2)$ for $c_N = 1$ or -1, and perform a rejection sampling over $z = \lfloor y \rceil$ with acceptance rate $\exp(-Y''/(2\sigma^2))$ where $Y'' = (\lfloor y \rceil + c_F)^2 - (y \mp 1)^2$ (we also need to ensure $Y'' \geq 0$ before performing this rejection sampling). The sampling algorithm for $\mathscr{D}_{-c_F,\sigma}$ is presented in Algorithm 5.2. Note that the output of Algorithm 5.2 is restricted to the domain $\mathbb{Z} \setminus \{0\}$. Therefore, the algorithm needs to output 0 with probability $\mathscr{D}_{-c_F,\sigma}(0)$. We present the full algorithm in Algorithm 5.3. Since both Algorithm 5.2 and Algorithm 5.3 do not require pre-computations related to σ, our scheme can support arbitrary standard deviations determined on-the-fly at run-time in addition to arbitrary centers.

Theorem 13. *The output z sampled by Algorithm 5.2 is distributed as $\mathscr{D}_{-c_F,\sigma}$ with domain $\mathbb{Z} \setminus \{0\}$. The output of Algorithm 5.3 is distributed as $\mathscr{D}_{c,\sigma}$ with domain $\mathbb{Z}$.*

Proof. When $b = 0$, y is distributed as $\mathcal{N}(-1, \sigma^2)$. For step 11 in Algorithm 5.2, we have $Y_1 = (\lfloor y \rceil + c_F)^2 - (y+1)^2 \geq 0$ for any $c_F \in [-1/2, 1/2]$ when $y \leq -1/2$. Therefore, the rejection condition $\exp(-Y_1/(2\sigma^2)) \in (0, 1]$. Let $z_0 = \lfloor y \rceil$. We have the output distribution:

$$\begin{aligned}\Pr[z = z_0] &\propto \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{(z_0+c_F)^2 - (y+1)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(z_0+c_F)^2}{2\sigma^2}\right) \mathrm{d}y = \rho_{-c_F,\sigma}(z_0). \end{aligned} \tag{1}$$

In this case, the distribution of $z = z_0$ is $\mathscr{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z}^-$ (due to the rejection of y to $(-\infty, -1/2]$).

Similarly, when $b = 1$, y is distributed as $\mathcal{N}(1, \sigma^2)$. For step 23 in Algorithm 5.2, we have $Y_2 = (\lfloor y \rceil + c_F)^2 - (y-1)^2 \geq 0$ for any $c_F \in [-1/2, 1/2]$ when $y \geq 1/2$. Therefore, the rejection condition $\exp(-Y_2/(2\sigma^2)) \in (0, 1]$. Let $z_0 = \lfloor y \rceil$. We have the output distribution:

$$\begin{aligned}\Pr[z = z_0] &\propto \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{(z_0+c_F)^2 - (y-1)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(z_0+c_F)^2}{2\sigma^2}\right) \mathrm{d}y = \rho_{-c_F,\sigma}(z_0). \end{aligned} \tag{2}$$

[2]Note that the shifting by ± 1 is not optimal. For the optimal shifting, please refer to the subsequent work [SZJ+21].

Algorithm 5.2 $\mathscr{D}_{-c_F,\sigma}$ sampler with domain $\mathbb{Z} \setminus \{0\}$.

Input: Center $c_F \in [-1/2, 1/2]$. Standard deviation $\sigma \in \mathbb{R}^+$.
Output: A sample z distributed as $\mathscr{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z} \setminus \{0\}$.

1: **function** RoundingSampler(c_F, σ)
2: Sample $x \hookleftarrow \mathscr{N}(0, 1)$.
3: Sample $b \hookleftarrow \mathscr{U}(\{0, 1\})$.
4: **if** $b = 0$ **then**
5: Let $y = \sigma \cdot x - 1$.
6: **if** $y > -1/2$ **then**
7: **goto** 2.
8: **end if**
9: Sample $r \hookleftarrow \mathscr{U}([0, 1))$.
10: Let $Y_1 = (\lfloor y \rceil + c_F)^2 - (y + 1)^2$.
11: **if** $r < \exp(-Y_1/(2\sigma^2))$ **then**
12: Let $z = \lfloor y \rceil$.
13: **else**
14: **goto** 2.
15: **end if**
16: **else**
17: Let $y = \sigma \cdot x + 1$.
18: **if** $y < 1/2$ **then**
19: **goto** 2.
20: **end if**
21: Sample $r \hookleftarrow \mathscr{U}([0, 1))$.
22: Let $Y_2 = (\lfloor y \rceil + c_F)^2 - (y - 1)^2$.
23: **if** $r < \exp(-Y_2/(2\sigma^2))$ **then**
24: Let $z = \lfloor y \rceil$.
25: **else**
26: **goto** 2.
27: **end if**
28: **end if**
29: **return** z.
30: **end function**

Algorithm 5.3 $\mathscr{D}_{c,\sigma}$ sampler with domain $\mathbb{Z}$.

Input: Center $c \in \mathbb{R}$. Standard deviation $\sigma \in \mathbb{R}^+$. Normalisation factor $S = \rho_{c,\sigma}(\mathbb{Z}) \approx \sigma\sqrt{2\pi}$.
Output: A sample distributed as $\mathscr{D}_{c,\sigma}(\mathbb{Z})$.

1: **function** RoundingSamplerFull(c, σ)
2: Let $c_I = \lfloor c \rceil$ and $c_F = c_I - c$.
3: Sample $r \hookleftarrow \mathscr{U}([0, 1))$.
4: **if** $r < \exp(-c_F^2/(2\sigma^2))/S$ **then**
5: Let $z' = 0$.
6: **else**
7: Let $z' =$ RoundingSampler(c_F, σ).
8: **end if**
9: **return** $z' + c_I$.
10: **end function**

In this case, the distribution of $z = z_0$ is $\mathscr{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z}^+$ (due to the rejection of y to $[1/2, \infty)$). Therefore, the output z in Algorithm 5.2 is distributed as $\mathscr{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z} \setminus \{0\}$.

In Algorithm 5.3, the probability $\Pr[z' = 0] = \exp(-c_F^2/(2\sigma^2))/S = \mathscr{D}_{-c_F,\sigma}(0)$. Therefore, variable z' is distributed as $\mathscr{D}_{-c_F,\sigma}$ with domain $\mathbb{Z}$. Since $c = c_I - c_F$, we have the output $z' + c_I$ distributed as $\mathscr{D}_{c,\sigma}$ with domain $\mathbb{Z}$. □

To prove the rejection rate of Algorithm 5.2, we need the following lemma:

Lemma 3. *For any $\epsilon \in (0,1)$ and $c \in [-1/2, 1/2]$, if $\sigma \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$, then both $\rho_{c,\sigma}(\mathbb{Z}^-)$ and $\rho_{c,\sigma}(\mathbb{Z}^+)$ have the lower bound: $\frac{1}{2} \cdot \frac{1-\epsilon}{1+\epsilon} \cdot \rho_\sigma(\mathbb{Z}) - 1$.*

Proof. When $c \in [-1/2, 1/2]$, for $\rho_{c,\sigma}(\mathbb{Z}^-)$, we have:

$$\rho_{c,\sigma}(\mathbb{Z}) = \rho_{c,\sigma}(\mathbb{Z}^+) + \rho_{c,\sigma}(\mathbb{Z}^- \cup \{0\}) \leq 2\rho_{c,\sigma}(\mathbb{Z}^- \cup \{0\}) = 2\rho_{c,\sigma}(\mathbb{Z}^-) + 2\rho_{c,\sigma}(0).$$

Therefore,

$$\begin{aligned}\rho_{c,\sigma}(\mathbb{Z}^-) &\geq \frac{1}{2} \cdot \rho_{c,\sigma}(\mathbb{Z}) - \rho_{c,\sigma}(0) \\ &\geq \frac{1}{2} \cdot \frac{1-\epsilon}{1+\epsilon} \cdot \rho_\sigma(\mathbb{Z}) - \rho_{c,\sigma}(0) \quad \text{(By Theorem 1).}\end{aligned}$$

We have $\rho_\sigma(0) \geq \rho_{c,\sigma}(0)$ for $c \in [-1/2, 1/2]$. Therefore,

$$\rho_{c,\sigma}(\mathbb{Z}^-) \geq \frac{1}{2} \cdot \frac{1-\epsilon}{1+\epsilon} \cdot \rho_\sigma(\mathbb{Z}) - 1.$$

Similarly, when $c \in [-1/2, 1/2]$, for $\rho_{c,\sigma}(\mathbb{Z}^+)$, we have:

$$\rho_{c,\sigma}(\mathbb{Z}) = \rho_{c,\sigma}(\mathbb{Z}^-) + \rho_{c,\sigma}(\mathbb{Z}^+ \cup \{0\}) \leq 2\rho_{c,\sigma}(\mathbb{Z}^+ \cup \{0\}) = 2\rho_{c,\sigma}(\mathbb{Z}^+) + 2\rho_{c,\sigma}(0).$$

Therefore, since $c \in [-1/2, 1/2]$, we have:

$$\begin{aligned}\rho_{c,\sigma}(\mathbb{Z}^+) &\geq \frac{1}{2} \cdot \rho_{c,\sigma}(\mathbb{Z}) - \rho_{c,\sigma}(0) \\ &\geq \frac{1}{2} \cdot \frac{1-\epsilon}{1+\epsilon} \cdot \rho_\sigma(\mathbb{Z}) - \rho_{c,\sigma}(0) \quad \text{(By Theorem 1)} \\ &\geq \frac{1}{2} \cdot \frac{1-\epsilon}{1+\epsilon} \cdot \rho_\sigma(\mathbb{Z}) - 1 \quad (\rho_\sigma(0) \geq \rho_{c,\sigma}(0) \text{ when } c \in [-1/2, 1/2]).\end{aligned}$$

□

Theorem 14. *For $\sigma \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$, the expected number of trials M in Algorithm 5.2 has the upper bound: $M \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{2\pi} - 1 - 2 \cdot \frac{1+\epsilon}{1-\epsilon}}$. If σ is much greater than $\left(1 + 2 \cdot \frac{1+\epsilon}{1-\epsilon}\right)/\sqrt{2\pi}$, then $M \leq 2 \cdot (1 + \mathcal{O}(\epsilon) + \mathcal{O}(1/\sigma))$.*

Proof. By Theorem 13, when $b = 0$, we have the output probability density function $f(y) = \rho_{-c_F,\sigma}(\lfloor y \rceil)/\rho_{-c_F,\sigma}(\mathbb{Z}^-)$ and the input probability density function $g(y) = \rho_{-1,\sigma}(y)/(\sigma\sqrt{2\pi})$. The expected number of trials can be written as:

$$M = \max \frac{f(y)}{g(y)} = \max \left(\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{-1,\sigma}(y)} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^-)} \right).$$

We have:

$$\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{-1,\sigma}(y)} = \frac{\exp\left(-\frac{(\lfloor y \rceil + c_F)^2}{2\sigma^2}\right)}{\exp\left(-\frac{(y+1)^2}{2\sigma^2}\right)} = \exp\left(-\frac{(\lfloor y \rceil + c_F)^2 - (y+1)^2}{2\sigma^2}\right) \leq 1.$$

Therefore,

$$M \leq \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^-)} \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_\sigma(\mathbb{Z}) - 2 \cdot \frac{1+\epsilon}{1-\epsilon}} \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{2\pi} - 1 - 2 \cdot \frac{1+\epsilon}{1-\epsilon}},$$

where the second inequality follows from Lemma 3, and the third inequality follows from $\rho_\sigma(\mathbb{Z}) = \rho_\sigma(\mathbb{Z}^- \cup \{0\}) + \rho_\sigma(\mathbb{Z}^+ \cup \{0\}) - 1$ and the sum-integral comparison: $\rho_\sigma(\mathbb{Z}^- \cup \{0\}) \geq \int_{-\infty}^{0} \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{\pi/2}$ and $\rho_\sigma(\mathbb{Z}^+ \cup \{0\}) \geq \int_{0}^{\infty} \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{\pi/2}$.

Similarly, when $b = 1$, we have the output probability density function $f(y) = \rho_{-c_F,\sigma}(\lfloor y \rceil)/\rho_{-c_F,\sigma}(\mathbb{Z}^+)$ and the input probability density function $g(y) = \rho_{1,\sigma}(y)/(\sigma\sqrt{2\pi})$. The expected number of trials can be written as:

$$M = \max \frac{f(y)}{g(y)} = \max \left(\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{1,\sigma}(y)} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^+)} \right).$$

We have:

$$\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{1,\sigma}(y)} = \frac{\exp\left(-\frac{(\lfloor y \rceil + c_F)^2}{2\sigma^2}\right)}{\exp\left(-\frac{(y-1)^2}{2\sigma^2}\right)} = \exp\left(-\frac{(\lfloor y \rceil + c_F)^2 - (y-1)^2}{2\sigma^2}\right) \leq 1.$$

Therefore,

$$M \leq \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^+)} \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_\sigma(\mathbb{Z}) - 2 \cdot \frac{1+\epsilon}{1-\epsilon}} \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{2\pi} - 1 - 2 \cdot \frac{1+\epsilon}{1-\epsilon}},$$

where the second inequality follows from Lemma 3, and the third inequality follows from $\rho_\sigma(\mathbb{Z}) \geq \sigma\sqrt{2\pi} - 1$.

When σ is much greater than $\left(1 + 2 \cdot \frac{1+\epsilon}{1-\epsilon}\right) \Big/ \sqrt{2\pi}$, $\sigma\sqrt{2\pi}$ is much greater than $1 + 2 \cdot \frac{1+\epsilon}{1-\epsilon}$. Thus,

$$M \leq 2 \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{2\pi} - 1 - 2 \cdot \frac{1+\epsilon}{1-\epsilon}} \leq 2 \cdot (1 + \mathcal{O}(\epsilon) + \mathcal{O}(1/\sigma)).$$

□

5.2 Accuracy Analysis

We now analyse the relative error of Algorithm 5.2 here. Let the absolute error of the continuous Gaussian sample x be e_x: $x' = x + e$, where x' is the actual sample, x is the ideal sample, and the error $|e| \leq e_x$. We denote the actual distribution by $\mathscr{P}_{\text{actual}}$ and the ideal distribution by $\mathscr{P}_{\text{ideal}}$. Since the variable y might be rounded to an incorrect integer due to the error from x when y is close to the boundaries $z_0 \pm 1/2$ [HLS18], we have:

$$\begin{aligned}
\Delta(\mathscr{P}_{\text{actual}} \| \mathscr{P}_{\text{ideal}}) &= \max \left| \frac{\mathscr{P}_{\text{actual}}}{\mathscr{P}_{\text{ideal}}} - 1 \right| \\
&= \max_{z_0} \left| \frac{\int_{z_0-1/2-\sigma e_x}^{z_0+1/2+\sigma e_x} \exp\left(-\frac{(z_0+c_F)^2}{2\sigma^2}\right) \mathrm{d}y}{\rho_{-c_F,\sigma}(z_0)} - 1 \right| \quad \text{(by (1), (2), and } y = \sigma x \pm 1) \\
&= \max_{z_0} \left| \frac{(1 + 2\sigma e_x) \cdot \rho_{-c_F,\sigma}(z_0)}{\rho_{-c_F,\sigma}(z_0)} - 1 \right| = 2\sigma e_x.
\end{aligned}$$

By Theorem 6, for λ-bit security, we need:

$$\begin{aligned}
R_{2\lambda}(\mathscr{P}_{\text{actual}} \| \mathscr{P}_{\text{ideal}}) \leq 1 + \frac{1}{4M} &\implies 1 + 2\lambda \cdot \frac{(\Delta(\mathscr{P}_{\text{actual}} \| \mathscr{P}_{\text{ideal}}))^2}{2} \leq 1 + \frac{1}{4M} \\
&\implies e_x \leq \frac{1}{4\sigma\sqrt{\lambda M}}.
\end{aligned}$$

Note that both $\mathscr{P}_{\text{actual}}$ and $\mathscr{P}_{\text{ideal}}$ have the same normalisation factor, since $\mathscr{P}_{\text{actual}}$ is obtained by the imperfect continuous Gaussian distribution with the rounding error contributed to the interval of the integral [HLS18].

5.3 Precision Analysis

To avoid sampling a uniformly random real r with high absolute precision at rejection steps 11 and 23 in Algorithm 5.2, and step 4 in Algorithm 5.3, we adapt the comparison approach similar to the FACCT in Chapter 4. Assume an IEEE-754 floating-point value $f \in (0, 1)$ with $(\delta_f + 1)$-bit precision is represented by $f = (1 + \mathit{mantissa} \cdot 2^{-\delta_f}) \cdot 2^{\mathit{exponent}}$, where integer $\mathit{mantissa}$ has δ_f bits and $\mathit{exponent} \in \mathbb{Z}^-$. To check $r < f$, one can sample

$r_m \hookleftarrow \mathcal{U}(\{0,1\}^{\delta_f+1})$, $r_e \hookleftarrow \mathcal{U}(\{0,1\}^{\ell})$, and check $r_m < mantissa + 2^{\delta_f}$ and $r_e < 2^{\ell+exponent+1}$ instead for some ℓ such that $\ell + exponent + 1 \geq 0$.

Here, we analyse the precision requirement of r_e. We have the following theorem for the worst-case acceptance rate in Algorithm 5.2:

Theorem 15. *Assume $x \in [-\tau, \tau]$ and $y \in [-\tau\sigma - 1, \tau\sigma + 1]$. In worst case, step 11 in Algorithm 5.2 has the acceptance rate:*

$$p_1 \geq \exp\left(-\frac{(-2\tau\sigma + c_F - 3/2)(c_F - 3/2)}{2\sigma^2}\right),$$

and step 23 in Algorithm 5.2 has the acceptance rate:

$$p_2 \geq \exp\left(-\frac{(2\tau\sigma + c_F + 3/2)(c_F + 3/2)}{2\sigma^2}\right).$$

Proof. For $b = 0$ and $y \leq -1/2$, we have the acceptance rate $p_1 = \exp(-Y_1/(2\sigma^2))$ at step 11 in Algorithm 5.2 where:

$$\begin{aligned}
Y_1 &= (\lfloor y \rceil + c_F)^2 - (y+1)^2 \\
&= (y + \delta + c_F)^2 - (y+1)^2 \quad (\lfloor y \rceil = y + \delta \text{ where } \delta \in [-1/2, 1/2]) \\
&= (2y + \delta + c_F + 1)(\delta + c_F - 1) \\
&\leq (-2\tau\sigma + c_F - 3/2)(c_F - 3/2) \quad (\text{when } \delta = -1/2 \text{ and } y = -\tau\sigma - 1).
\end{aligned}$$

Similarly, for $b = 1$ and $y \geq 1/2$, we have the acceptance rate $p_2 = \exp(-Y_2/(2\sigma^2))$ at step 23 in Algorithm 5.2 where:

$$\begin{aligned}
Y_2 &= (\lfloor y \rceil + c_F)^2 - (y-1)^2 \\
&= (y + \delta + c_F)^2 - (y-1)^2 \quad (\lfloor y \rceil = y + \delta \text{ where } \delta \in [-1/2, 1/2]) \\
&= (2y + \delta + c_F - 1)(\delta + c_F + 1) \\
&\leq (2\tau\sigma + c_F + 3/2)(c_F + 3/2) \quad (\text{when } \delta = 1/2 \text{ and } y = \tau\sigma + 1).
\end{aligned}$$

□

Let $\Delta \leq 1/2$ be the maximum relative error of the right hand side computations at rejection steps 11 and 23 in Algorithm 5.2, and step 4 in Algorithm 5.3. For $\exp(-Y_1/(2\sigma^2))$

at step 11 in Algorithm 5.2, we have:

$$\begin{aligned} exponent_1 &\geq \lfloor \log_2((1-\Delta)\cdot\exp(-Y_1/(2\sigma^2)))\rfloor \\ &\geq \left\lfloor -1 - \frac{(-2\tau\sigma + c_F - 3/2)(c_F - 3/2)}{2\sigma^2}\cdot\log_2 e \right\rfloor \quad \text{(by Theorem 15 and } \Delta \leq 1/2) \\ &\geq \left\lfloor -1 - \frac{2\tau\sigma+2}{\sigma^2}\cdot\log_2 e \right\rfloor \quad \text{(when } c_F = -1/2). \end{aligned}$$

Similarly, for $\exp(-Y_2/(2\sigma^2))$ at step 23 in Algorithm 5.2, we have:

$$\begin{aligned} exponent_2 &\geq \lfloor \log_2((1-\Delta)\cdot\exp(-Y_2/(2\sigma^2)))\rfloor \\ &\geq \left\lfloor -1 - \frac{(2\tau\sigma + c_F + 3/2)(c_F + 3/2)}{2\sigma^2}\cdot\log_2 e \right\rfloor \quad \text{(by Theorem 15 and } \Delta \leq 1/2) \\ &\geq \left\lfloor -1 - \frac{2\tau\sigma+2}{\sigma^2}\cdot\log_2 e \right\rfloor \quad \text{(when } c_F = 1/2). \end{aligned}$$

For $\exp(-c_F^2/(2\sigma^2))/S$ at step 4 in Algorithm 5.3, we have:

$$\begin{aligned} exponent_3 &\geq \lfloor \log_2((1-\Delta)\cdot\exp(-c_F^2/(2\sigma^2))/S)\rfloor \\ &\geq \left\lfloor -1 - \frac{1}{8\sigma^2}\cdot\log_2 e - \log_2(\sigma\sqrt{2\pi}) \right\rfloor \quad \text{(when } c_F = \pm 1/2 \text{ and } \Delta \leq 1/2). \end{aligned}$$

Therefore, we have:

$$exponent \geq \min\left\{ \left\lfloor -1 - \frac{2\tau\sigma+2}{\sigma^2}\cdot\log_2 e \right\rfloor, \left\lfloor -1 - \frac{1}{8\sigma^2}\cdot\log_2 e - \log_2(\sigma\sqrt{2\pi}) \right\rfloor \right\}.$$

Since the probability $\Pr[-\tau \leq x \leq \tau] = \mathrm{erf}(\tau/\sqrt{2})$ for $x \hookleftarrow \mathcal{N}(0,1)$, to ensure $1 - \Pr[-\tau \leq x \leq \tau] \leq 2^{-\lambda}$, we need $\tau \geq \sqrt{2}\cdot\mathrm{erf}^{-1}(1-2^{-\lambda})$. Therefore, for $\lambda = 128$ and $\sigma \in [2, 2^{20}]$, we have $\tau \geq 13.11$, $exponent \geq -23$, and thus $\ell \geq 22$, i.e. r_e needs to have at least 22 bits.

5.4 Evaluation

Side-channel Resistance Our implementation is not fully constant-time because the rejection rate may still reveal σ due to Theorem 14. However, since the rejection rate is independent of the center, our implementation can achieve fully constant-time with respect to the secret if σ is public. The σ in convolution-style lattice trapdoor samplers [MP12, Pei10] is typically a public constant, but σ in GPV-style sampler [GPV08] depends on the secret. Note that the IBE implementation from [BFRS18] adapted a variant of [MP12], but it appears that the implementation source code[3] of [BFRS18] used a different distribution and the side-channel resistance perspective is unclear. Our sampling

[3] https://github.com/lbibe/code.

algorithm can be applied in the IBE implementation of [BFRS18] to give a fully constant-time IBE implementation.

We perform benchmarks of Algorithm 5.3 with fixed σ and random arbitrary centers. We employ the Box-Muller continuous Gaussian sampler [HLS18, ZCHW17] implemented by using the `VCL` library [Fog17], which provides $e_x \leq 2^{-48}$ [HLS18]. To compare with [MR18], we select $\sigma = \{2, 4, 8, 16, 32\}$, and to compare with [MW17], we choose $\sigma = 2^{15}$. In addition, we also compare with the FACCT in Chapter 4 and the variant [DWZ19] of the binary sampling algorithm [DDLL13] for additional $\sigma = \{2^{17}, 2^{20}\}$. From the error analysis in Section 5.2, for given e_x and λ: $M \leq \frac{1}{16\lambda e_x^2 \sigma^2}$. For $\sigma \in [2, 2^{20}]$ and $\lambda = 128$, we have $M \leq 2^{45}$. We adapt techniques similar to the FACCT in Chapter 4 to avoid high precision arithmetic (see Section 5.3 for details) and the scheme is implemented[4] by using the double precision i.e. $\delta_f = 52$. We also compute the normalisation factor S in double precision. We use the AES256 counter mode with hardware AES instructions (AES-NI) [Gue09] to generate the randomness in our implementations. We provide both the non-constant time reference implementation and the center-independent run-time implementation. We take care of all the branches for the center-independent run-time implementation by adapting constant-time selection techniques [Aum19]. For the non-constant time reference implementation (the "Reference" column in Table 5.1), we use the $\exp(x)$ from the C library, which provides about 50-bit precision [PFH+17], while for the center-independent run-time implementation (the "Center-independent" column in Table 5.1), we adapt the FACCT techniques from Chapter 4 with about 45-bit precision. From the precision analysis from [Pre17] and Chapter 4, the above precisions (including the precision of S) are sufficient for $\lambda = 128$ and $M \leq 2^{45}$.

The benchmark is carried on as follows: we use g++ 9.1.1 to compile our implementations with the compiling options `-O3 -march=native` enabled. The benchmark is running on an Intel i7-7700K CPU at 4.2GHz, with the Hyperthreading and the Turbo Boost disabled. We generate 1024 samples (with a random arbitrary center per sample) for 1000 times and measure the consumed CPU cycles, with the exception that we fix $c = 0$ and compare our center-independent run-time implementation with the FACCT in Chapter 4, since the FACCT is essentially a constant-time zero-centered discrete Gaussian sampler. Then, we convert the CPU cycles to the average number of samples per second for the comparison purpose with previous works.

The benchmark results of our scheme are shown in Table 5.1 (in the format of mean $\pm$ standard deviation). We also summarise the performance of previous works in Table 5.2, and show the comparison with the FACCT in Table 5.4 when $c = 0$. Since previous works [DWZ19, MR18, MW17] measured the number of generated samples per second running

[4] Our implementation is available at https://gitlab.com/raykzhao/gaussian_ac.

on CPUs with different frequencies, we scale all the numbers to be based on 4.2GHz.[5] In addition, since some previous works [MR18, MW17] require pre-computations to implement the sampling schemes, we summarise the pre-computation memory storage consumption in Table 5.3.[6] Because the TwinCDT method [MR18] provided different tradeoffs between the run-time speed and the pre-computation storage consumption, we show all 3 different sampling speeds and the corresponding pre-computation storage consumption for each σ from [MR18]. Note that although our sampling scheme does not require pre-computations, however, the $\exp(x)$ implementation typically consumes a small amount of memory to store the coefficients of the polynomial approximation. For example, the polynomial approximation of the $\exp(x)$ in our center-independent run-time implementation (adapted from the FACCT in Chapter 4) has degree 10 with double precision coefficients, and therefore it consumes $(10 + 1) \cdot 8 = 88$ bytes.

Table 5.1: Number of Samples per Second for Our Scheme with Fixed σ at 4.2GHz (with $\lambda = 128$).

σ	Reference ($\times 10^6$)	Center-independent ($\times 10^6$)
2	10.33 ± 0.18	8.96 ± 0.16
4	11.57 ± 0.18	10.87 ± 0.15
8	11.95 ± 0.17	11.61 ± 0.13
16	12.14 ± 0.16	12.00 ± 0.12
32	12.19 ± 0.15	12.21 ± 0.11
2^{15}	11.70 ± 0.13	11.57 ± 0.09
2^{17}	11.20 ± 0.14	11.63 ± 0.10
2^{20}	11.17 ± 0.13	11.28 ± 0.09

Table 5.2: Summary of the Speed of Previous Works for Fixed σ at 4.2GHz (with $\lambda = 128$).

σ	Number of Samples ($\times 10^6$/sec)
2 [MR18]	51.01/62.45/76.43
4 [MR18]	45.50/56.44/69.09
8 [MR18]	37.70/53.31/63.51
16 [MR18]	31.29/37.63/52.29
32 [MR18]	34.38/39.76/42.60
2^{15} [MW17]	$\approx$ 12.35 (online), 1.78 (online+offline)
4–2^{20} [DWZ19]	$\approx$ 16.3

From Table 5.1, our scheme has good performance for both small and large σ (11.53×10^6 samples per second for the non-constant time reference implementation and 11.27×10^6 samples per second for the center-independent run-time implementation on average).

[5] The online benchmark result from [MW17] is based on the authors' reference implementation, which is not claimed to be optimal. In addition, the online+offline benchmark result of [MW17] is obtained and scaled from the variant implemented by [DWZ19].

[6] The base sampler and the Bernoulli sampler of [DWZ19] may require pre-computations depending on the implementation techniques.

Table 5.3: Summary of the Storage of Previous Works for Fixed σ at 4.2GHz (with $\lambda = 128$).

σ	Pre-computation Storage (KB)
2 [MR18]	1.4/4.6/46
4 [MR18]	1.9/6.3/63
8 [MR18]	3/10/100
16 [MR18]	5.2/17/172
32 [MR18]	9.5/32/318
2^{15} [MW17]	$2^{5.4}$
4–2^{20} [DWZ19]	–

Table 5.4: Number of Samples per Second Compared with the FACCT for Fixed σ and $c = 0$ at 4.2GHz (with $\lambda = 128$).

σ	COSAC ($\times 10^6$/sec)	FACCT ($\times 10^6$/sec)
2	9.44	19.87
4	11.10	19.04
8	12.08	19.04
16	12.63	18.62
32	12.93	18.80
2^{15}	12.67	18.36
2^{17}	12.67	18.90
2^{20}	13.04	18.70

In particular, our scheme has better performance for large σ since the number of trials becomes lower by Theorem 14. Note that the amount of randomness required by the comparison steps in Section 5.3 will significantly increase for very small or very large σ. Therefore, our implementation consumes different amount of randomness in comparison steps for each σ based on Section 5.3, and the performance for some larger σ is slightly slower than smaller σ in Table 5.1 due to the increased amount of randomness required. The overhead introduced by the center-independent run-time implementation is at most 13.33% in our benchmarks. Note that the overhead of the center-independent run-time implementation is smaller for large σ due to the lower probability of outputting $z' = 0$ in Algorithm 5.3.

For $\sigma \in [2, 32]$, although the TwinCDT method [MR18] is 2.5x–7.3x faster than our non-constant time reference implementation, however, this method requires a pre-computation with at least 1.4 KB memory consumption to store the CDT, while our scheme only requires at most several hundred bytes if considering all the polynomial approximation coefficients (including those functions used by the Box-Muller continuous Gaussian sampler). When scaling up σ, the TwinCDT method [MR18] also costs much larger amount of memory (the pre-computation storage size increases by a factor of 6.7–6.9 when σ changes from 2 to 32), and the performance becomes significantly worse (the number of samples per second decreases by 32.6–44.3% when σ changes from

2 to 32). On the contrary, the pre-computation storage of our scheme is independent of σ and only relies on the precision requirements. Our scheme is also scalable and maintains good performance even for large $\sigma = 2^{15}$. In addition, for applications sampling from various σ such as the DLP IBE scheme [DLP14], one sampler subroutine implemented by using our scheme is able to serve all σ since the implementation does not require any pre-computations depending on σ, while the TwinCDT method [MR18] needs to pre-compute a different CDT for each σ.

Compared with [MW17] for $\sigma = 2^{15}$, if we measure both the online and offline phase run-time speed in total, our center-independent run-time implementation[7] achieves better performance in terms of both timing (6.5x faster) and pre-computation storage (the reference implementation from [MW17] requires about 42 KB to implement the Knuth-Yao offline batch sampler). The online-phase only run-time speed in [MW17] is slightly (1.07x) faster than our scheme. On the other hand, our scheme requires no offline pre-computations related to a specific discrete Gaussian distribution. In addition, our scheme can also be accelerated if we generate all the continuous Gaussian samples during the offline phase and only perform the rejection during the online phase. In this case, our center-independent run-time implementation generates 13.73×10^6 samples per second during the online phase, which is 1.11x faster than [MW17].

For the comparison with variants of the binary sampling algorithm, in Table 5.2, our non-constant time reference implementation is about 28.2% slower than [DWZ19] for $\sigma \in [4, 2^{20}]$ with arbitrary centers, and from Table 5.4, our center-independent run-time implementation is 30.3%–52.5% slower than the FACCT when $c = 0$ and $\sigma \in [2, 2^{20}]$. However, the FACCT scheme in Chapter 4 does not support an arbitrary center, while the side-channel resistance perspective of [DWZ19] is unclear. We expect that our implementation can achieve at most about 73.5% of the run-time speed of [DWZ19] and the FACCT on average for large σ, since both binary sampling variants require less than 1.47 trials per sample on average, while the average number of trials per sample is close to 2 in our scheme for large σ.

In addition, the performance of our scheme heavily relies on the underlying continuous Gaussian sampling algorithm. Currently the Box-Muller continuous Gaussian sampler [HLS18, ZCHW17] employed by our scheme is implemented by using the general purpose arithmetic routines (notably sqrt, ln, sin, and cos) from the `VCL` library [Fog17]. However, if lower precision is sufficient for the application, to achieve better performance in terms of speed and memory consumption, one may replace these general purpose arithmetic routines with hand-optimised code tailored for the target precision. To

[7] Here, we compare the performance with our center-independent run-time implementation because the implementation from [MW17] is constant-time.

realise this, the polynomial approximation techniques discussed in Chapter 4 might be handy.

5.5 Research Impact

Very recently, a variant of the COSAC sampler [SZJ+21] further reduced the average number of trials from 2 to nearly 1 per sample. Instead of shifting the centers of samples from the continuous Gaussian distribution by 1, this variant shifts the centers by c_F from the input. The average number of trials is also center-independent in this sampler. In practice, the sampler implementation from [SZJ+21] achieves a 1.46x–1.63x speedup compared to our COSAC implementation for $\sigma \in [2, 2^{20}]$ during the benchmarks.

In addition, very recently, Aranha et al. employed our COSAC sampler in the implementation of their lattice-based electronic voting scheme [ABG+21].

Chapter 6

Lattice-based HIBE (Latte)

This chapter was *partially* stored in the preprint:

- Raymond K. Zhao, Sarah McCarthy, Ron Steinfeld, Amin Sakzad, and Máire O'Neill. Quantum-safe HIBE: does it cost a Latte?. (2021). IACR Cryptology ePrint Archive: Report 2021/222.

This chapter provides the first performance benchmarking of a quantum-safe HIBE scheme, Latte, written in C.[1] We also identify bottlenecks, propose optimisations for Latte and consider its suitability for such applications. In more detail, the contributions of this chapter are:

- We adapt the FFT sampling procedures from Falcon [PFH+17], which is faster than the Klein-GPV sampler [Kle00, GPV08] used in the original Latte technical report [ETS19]. In addition, the proposed Latte specification [ETS19] did not discuss the integer discrete Gaussian sampling techniques suitable for the needed standard deviations. We integrate efficient sampling techniques including the FACCT in Chapter 4 and the variant [SZJ+21] of the COSAC techniques in Chapter 5 in our optimised Latte implementation.

- For the (Mod)NTRU basis in Latte, we provide an optimised ffLDL algorithm [DP16, PFH+17], which is used by the FFT sampling procedure [PFH+17]. We observe that for the (Mod)NTRU basis in Latte, the computation of $\mathbf{D}$ in the ffLDL algorithm [DP16, PFH+17] when performing the $\mathbf{LDL}^*$ decomposition in the FFT domain can be done by solely using the real number arithmetic without complex number arithmetic. By adapting this observation, under 256-bit floating-point arithmetic precision, our optimised ffLDL implementation achieves a 71.1%–73.4%

[1] The latest implementation source code is available at https://gitlab.com/raykzhao/latte.

speedup on average compared to a naive generic ffLDL implementation for the (Mod)NTRU basis with the Latte parameter sets.

- We provide the provable theoretical error analysis for our optimised ffLDL algorithm in Latte. This is the first provable concrete error analysis of the ffLDL algorithm, with only a couple of mild and explicitly stated heuristics, since Falcon [PFH+17] only provided the heuristic error bounds based on experimental results with very little technical discussion. based on the Latte parameter sets and show that for (Mod)NTRU basis in the Latte scheme, both the *relative* error of the standard deviation σ (i.e. σ used by the integer discrete Gaussian sampling subroutine in the FFT sampling procedure) at leaves of the ffSampling tree and the *absolute* error of $\mathbf{L}$ (i.e. values at non-leaf nodes in the tree) in the ffLDL output are linearly proportional to the floating-point arithmetic precision.

- We adapt the NTRUSolve function from Falcon [PFH+17] in order to efficiently solve the NTRU equation in our optimised Latte KeyGen algorithm. The NTRUSolve is both faster and more compact [PP19] than the resultant method [HHP+03, DLP14] used in the original Latte technical report [ETS19]. In addition, we adapt the technique from ModFalcon [CPS+20] and the length reduction technique by using Cramer's rule [ETS19] in order to efficiently solve the NTRU equation for higher lattice dimensions in our optimised Latte Delegate algorithm.

- We give the first complete performance results of a lattice-based HIBE scheme, including the KeyGen, Delegate, and Extract algorithms, for which the implementation results were unclear in the original Latte technical report [ETS19]. This technical report estimated that Delegate would have run-time in the order of minutes on a desktop machine. Using the optimised techniques in this chapter, we show that an efficient implementation can perform the Delegate function in only a few seconds on a desktop machine.

6.1 Latte Software Design Features and Considerations

6.1.1 Techniques from Falcon and ModFalcon

Our optimised software design of Latte utilises techniques from the digital signature scheme Falcon [PFH+17]. The two schemes are closely related; they are instantiated over the same type of lattice and share key generation and sampling procedures. Falcon makes use of the "tower of rings" structure to find a solution to the NTRU equation $\mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$, for a given $\mathbf{f}$ and $\mathbf{g}$ in the NTRUSolve sub-algorithm of

KeyGen. The tower of rings approach utilises the fact that computations over polynomials $\mathbf{f}, \mathbf{g} \in \mathbb{C}[x]/\langle x^{N/2}+1\rangle$ are equivalent to computations over $\mathbf{f}(x^2), \mathbf{g}(x^2) \in \mathbb{C}[x]/\langle x^N+1\rangle$. When $N = 2^k$, for some $k \in \mathbb{Z}$, this can be applied repeatedly so that computations are performed over polynomials of degree 1. This brings advantages in terms of both memory usage and speed [PP19]. Therefore, we replace the resultant method [HHP+03, DLP14] of solving the NTRU equation in the original Latte KeyGen (Line 7–14 in Algorithm 3.8) with the NTRUSolve function [PFH+17, PP19] (see Algorithm 6.1) in our optimised Latte KeyGen algorithm (Line 7–10 in Algorithm 6.2). $N(\mathbf{f}), N(\mathbf{g})$ at Line 10–11 in Algorithm 6.1 are the field norms of $\mathbf{f}, \mathbf{g} \in \mathbb{Z}[x]/\langle x^n+1\rangle$ as defined in [PP19].

Algorithm 6.1 $\text{NTRUSolve}_{N,q}$ [PFH+17, PP19].

Input: $\mathbf{f}, \mathbf{g} \in \mathbb{Z}[x]/\langle x^N+1\rangle$.
Output: $\mathbf{F}, \mathbf{G} \in \mathbb{Z}[x]/\langle x^N+1\rangle$ such that $\mathbf{fG} - \mathbf{gF} = q \mod x^N+1$.

1: **function** $\text{NTRUSolve}_{N,q}(\mathbf{f}, \mathbf{g})$
2: **if** $N = 1$ **then**
3: Compute $u, v \in \mathbb{Z}$ such that $u\mathbf{f} - v\mathbf{g} = \gcd(\mathbf{f}, \mathbf{g})$.
4: **if** $\gcd(\mathbf{f}, \mathbf{g}) \neq 1$ **then**
5: abort.
6: **end if**
7: $(\mathbf{F}, \mathbf{G}) \leftarrow (vq, uq)$.
8: **return** $(\mathbf{F}, \mathbf{G})$.
9: **else**
10: $\mathbf{f}' \leftarrow N(\mathbf{f})$.
11: $\mathbf{g}' \leftarrow N(\mathbf{g})$.
12: $(\mathbf{F}', \mathbf{G}') \leftarrow \text{NTRUSolve}_{N/2,q}(\mathbf{f}', \mathbf{g}')$.
13: $\mathbf{F} \leftarrow \mathbf{F}'(x^2) \cdot \mathbf{f}'(x^2)/\mathbf{f}(x)$.
14: $\mathbf{G} \leftarrow \mathbf{G}'(x^2) \cdot \mathbf{g}'(x^2)/\mathbf{g}(x)$.
15: $\mathbf{k} \leftarrow \left\lfloor \frac{\mathbf{F}\cdot\mathbf{f}^* + \mathbf{G}\cdot\mathbf{g}^*}{\mathbf{f}\cdot\mathbf{f}^* + \mathbf{g}\cdot\mathbf{g}^*} \right\rceil \in \mathfrak{R}$.
16: $\mathbf{F} \leftarrow \mathbf{F} - \mathbf{k}\cdot\mathbf{f}$ and $\mathbf{G} \leftarrow \mathbf{G} - \mathbf{k}\cdot\mathbf{g}$.
17: **return** $(\mathbf{F}, \mathbf{G})$.
18: **end if**
19: **end function**

In addition, to accelerate the lattice discrete Gaussian sampling, the Klein-GPV sampler [Kle00, GPV08] used in the original Latte scheme (Line 4–5 in Algorithm 3.9; Line 4–5 in Algorithm 3.10) is replaced by the FFT sampling procedures (see Section 2.2.5) from Falcon [PFH+17] in our optimised Latte Delegate and Extract algorithms (Line 3, 6–8 in Algorithm 6.3; Line 4–7 in Algorithm 6.4).

Furthermore, in Latte Delegate, to complete the delegated basis $\mathbf{S}_\ell$ for lattice dimension higher than $2N$, we adapt the technique from ModFalcon [CPS+20]. Let $\mathbf{S}_\ell = \begin{pmatrix} \mathbf{v}^\mathsf{T} & \mathbf{M} \\ \mathbf{G}_\ell & \mathbf{F}'_\ell \end{pmatrix}$ be the delegated basis, where $\mathbf{G}_\ell = \mathbf{s}_{\ell+1,0}$, $\mathbf{F}'_\ell = (\mathbf{s}_{\ell+1,1}, \ldots, \mathbf{s}_{\ell+1,\ell+1})$, $\mathbf{v} = (\mathbf{s}_{0,0}, \mathbf{s}_{1,0}, \ldots, \mathbf{s}_{\ell,0})$, and $\mathbf{M} = (\mathbf{s}_{i,j})$ for $0 \leq i \leq \ell$ and $1 \leq j \leq \ell+1$. By Schur complement, if $\mathbf{M}$ is invertible,

Algorithm 6.2 Optimised Latte KeyGen algorithm.

Input: N, q, σ_0.
Output: $\mathbf{S}_0 \in \mathfrak{R}^{2\times 2}, \mathbf{h}, \mathbf{b} \in \mathfrak{R}_q$.

1: **function** KeyGen
2: $\mathbf{f}, \mathbf{g} \hookleftarrow \mathscr{D}^N_{\sigma_0}$.
3: Norm $\leftarrow \max\left(\|\mathbf{g}, -\mathbf{f}\|, \left\|\left(\frac{q\cdot\mathbf{f}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}, \frac{q\cdot\mathbf{g}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}\right)\right\|\right)$.
4: **if** Norm $> \sigma_0 \cdot \sqrt{2N}$ **then**
5: **goto** Step 2.
6: **end if**
7: $\mathbf{F}, \mathbf{G} \leftarrow \text{NTRUSolve}_{N,q}(\mathbf{f}, \mathbf{g})$.
8: **if** NTRUSolve is aborted **then**
9: **goto** Step 2.
10: **end if**
11: $\mathbf{h} \leftarrow \mathbf{g} \cdot \mathbf{f}^{-1} \mod q$ in NTT domain.
12: $\mathbf{b} \hookleftarrow \mathscr{U}(\mathfrak{R}_q)$ in NTT domain.
13: **return** $\mathbf{S}_0 = \begin{pmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{pmatrix}, \mathbf{h}, \mathbf{b}$.
14: **end function**

we have:

$$\begin{aligned}\det(\mathbf{S}_\ell) &= \det(\mathbf{M}) \cdot \det(\mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \mathbf{M}^{-1} \cdot \mathbf{v}^\mathsf{T}) = \det(\mathbf{M}) \cdot (\mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \mathbf{M}^{-1} \cdot \mathbf{v}^\mathsf{T}) \\ &= \det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \mathrm{adj}(\mathbf{M}) \cdot \mathbf{v}^\mathsf{T}.\end{aligned}$$

Since one can choose any $(\mathbf{S}_\ell)_{l+1} = (\mathbf{G}_\ell, \mathbf{F}'_\ell)$ such that $\det(\mathbf{S}_\ell) = q$ when filling the bottom row $(\mathbf{S}_\ell)_{l+1}$ of $\mathbf{S}_\ell$, let $\mathbf{F}'_\ell$ have the form $(\mathbf{F}_\ell, \mathbf{0}, \ldots, \mathbf{0})$. We have $\det(\mathbf{S}_\ell) = \det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}_\ell \cdot \mathbf{u}_0$ where $\mathbf{u}_0$ is the first coordinate of $\mathbf{u} = \mathrm{adj}(\mathbf{M}) \cdot \mathbf{v}^\mathsf{T}$. In order to fill the bottom row $(\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1})$ of $\mathbf{S}_\ell$, if $\mathbf{M}$ is invertible, we can use the same NTRUSolve algorithm as in Latte KeyGen to find $\mathbf{F}_\ell, \mathbf{G}_\ell$ such that $\det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}_\ell \cdot \mathbf{u}_0 = q$, and we simply resample when $\det(\mathbf{M}) = 0$. The resultant approach in the original Latte Delegate (Line 11–19 in Algorithm 3.9) is replaced by the technique above in our optimised Latte Delegate algorithm (Line 13–22 in Algorithm 6.3).

However, since the NTRUSolve algorithm [PP19] performs the length reduction based on the size of the coefficients in the input, the coefficient size of the output $\mathbf{F}_\ell, \mathbf{G}_\ell$ will be approximately the coefficient size of the input $\det(\mathbf{M}), \mathbf{u}_0$. Since $\mathbf{M}$ is an $(\ell+1) \times (\ell+1)$ sub-matrix of $\mathbf{S}_\ell$ with size of coordinate in the order of q among each element, the size of coefficient in $\det(\mathbf{M}), \mathbf{u}_0$, and $\mathbf{F}_\ell, \mathbf{G}_\ell$ will be in the order of $q^{\ell+1}$. In order to adapt the estimation of $\|(\mathbf{S}_\ell)_{\ell+1}\|$ from [Pre15] when analysing the errors in Section 6.3.4, $(\mathbf{S}_\ell)_{\ell+1}$ must be fully length-reduced against $(\mathbf{S}_\ell)_i$ for $0 \leq i \leq \ell$. Therefore, we need further length reduction by using Cramer's rule from the original Latte Delegate algorithm (Line 20–25 in Algorithm 3.9).

Our optimised Latte Key Generation, Delegate, and Extract algorithms are shown in Algorithm 6.2, Algorithm 6.3, and Algorithm 6.4, respectively.

Algorithm 6.3 Optimised Latte Delegate algorithm (from level $\ell - 1$ to ℓ).

Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathfrak{R}_q, \mathsf{ID}_\ell$.
Output: $\mathbf{S}_\ell \in \mathfrak{R}^{(\ell+2)\times(\ell+2)}$.

1: **function** Delegate
2: $\mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1|\ldots|\mathsf{ID}_\ell)$ in NTT domain.
3: $T_{\ell-1} \leftarrow \mathrm{Tree}(\mathbf{S}_{\ell-1}, \sigma_\ell)$.
4: **for** $i = 0$ **to** ℓ **do**
5: $\mathbf{s}_{i,\ell+1} \hookleftarrow \mathscr{D}^N_{\sigma_\ell}$.
6: $\mathbf{t} \leftarrow (-\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \ldots, \mathbf{0}) \cdot \mathbf{S}^{-1}_{\ell-1}$.
7: $\mathbf{z} \leftarrow \mathrm{FFT}^{-1}(\mathrm{ffSampling}(\mathbf{t}, T_{\ell-1}))$.
8: $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,\ell}) \leftarrow (\mathbf{t} - \mathbf{z}) \cdot \mathbf{S}_{\ell-1}$.
9: **if** $\|(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,\ell}, \mathbf{s}_{i,\ell+1})\| > \sqrt{(\ell+2)N} \cdot \sigma_\ell$ **then**
10: Resample.
11: **end if**
12: **end for**
13: Set $\mathbf{M} = (\mathbf{s}_{i,j})$, for $0 \le i \le \ell$ and $1 \le j \le \ell + 1$.
14: **if** $\mathbf{M}$ is not invertible **then**
15: **goto** Step 4.
16: **end if**
17: $\mathbf{u} \leftarrow \mathrm{adj}(\mathbf{M}) \cdot (\mathbf{s}_{0,0}, \mathbf{s}_{1,0}, \ldots, \mathbf{s}_{\ell,0})^\mathsf{T}$.
18: $(\mathbf{F}_\ell, \mathbf{G}_\ell) \leftarrow \mathrm{NTRUSolve}_{N,q}(\det(\mathbf{M}), \mathbf{u}_0)$ where $\mathbf{u}_0$ is the first coordinate of $\mathbf{u}$.
19: **if** NTRUSolve is aborted **then**
20: **goto** Step 4.
21: **end if**
22: $(\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1}) \leftarrow (\mathbf{G}_\ell, \mathbf{F}_\ell, \mathbf{0}, \ldots, \mathbf{0})$.
23: Set $\mathbf{C} = (\mathbf{c}_{i,j})$, where $\mathbf{c}_{i,j} = \mathbf{s}_{j,0} \cdot \mathbf{s}^*_{i,0} + \cdots + \mathbf{s}_{j,\ell+1} \cdot \mathbf{s}^*_{i,\ell+1}$, $0 \le i, j \le \ell$.
24: Set $\mathbf{d} = (\mathbf{d}_i)$, where $\mathbf{d}_i = \mathbf{s}_{\ell+1,0} \cdot \mathbf{s}^*_{i,0} + \cdots + \mathbf{s}_{\ell+1,\ell+1} \cdot \mathbf{s}^*_{i,\ell+1}$, $0 \le i \le \ell$.
25: Let $\mathbf{k} = (\mathbf{k}_i)_{0 \le i \le \ell}$ be the solution to $\mathbf{C} \cdot \mathbf{k} = \mathbf{d}$. By Cramer's rule, $\mathbf{k}_i = \frac{\det(\mathbf{C}_i(\mathbf{d}))}{\det(\mathbf{C})}$, where $\mathbf{C}_i(\mathbf{d})$ is the matrix $\mathbf{C}$ with its i^{th} column replaced by $\mathbf{d}$.
26: **for** $i = 0$ **to** ℓ **do**
27: $(\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1}) = (\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1}) - \lfloor \mathbf{k}_i \rceil \cdot (\mathbf{s}_{i,0}, \ldots, \mathbf{s}_{i,\ell+1})$.
28: **end for**
29: **return** $\mathbf{S}_\ell = (\mathbf{s}_{i,j})$, for $0 \le i, j \le \ell + 1$.
30: **end function**

6.1.2 Discrete Gaussian Sampling over the Integers

In Latte KeyGen, $\mathbf{f}, \mathbf{g}$ may need to be resampled multiple times due to the norm check and possible failure to find solutions of the NTRU equation. In order to sample $2N$ coordinates efficiently from $\mathscr{D}_{\sigma_0}$, we employ the FACCT sampler in Chapter 4, which is fast and compact even for larger σ_0 used in Latte-3 and 4 parameter sets. However,

Algorithm 6.4 Optimised Latte Extract algorithm (from level $\ell - 1$ to user at level ℓ).

Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathbb{Z}_q^N, \mathsf{ID}_\ell$.
Output: $\mathbf{t}_0, \ldots, \mathbf{t}_{\ell+1} \in \mathfrak{R}_q$.

1: **function** Extract
2: $\quad \mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1 | \ldots | \mathsf{ID}_\ell)$ in NTT domain.
3: $\quad \mathbf{t}_{\ell+1} \hookleftarrow \mathscr{D}_{\sigma_\ell}^N$.
4: $\quad T_{\ell-1} \leftarrow \mathrm{Tree}(\mathbf{S}_{\ell-1}, \sigma_\ell)$.
5: $\quad \mathbf{t} \leftarrow (\mathbf{b} - \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \ldots, \mathbf{0}) \cdot \mathbf{S}_{\ell-1}^{-1}$.
6: $\quad \mathbf{z} \leftarrow \mathrm{FFT}^{-1}(\mathrm{ffSampling}(\mathbf{t}, T_{\ell-1}))$.
7: $\quad (\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_\ell) \leftarrow (\mathbf{t} - \mathbf{z}) \cdot \mathbf{S}_{\ell-1}$.
8: $\quad$ **return** $\mathbf{t}_0, \ldots, \mathbf{t}_{\ell+1} \in \mathfrak{R}_q$ in NTT domain.
9: **end function**

since the FACCT sampler can only sample with $\sigma = k\sqrt{1/(2\ln 2)}$ where k is a positive integer, we slightly increase $\sigma_0 \approx 1.17\sqrt{q/(2N)}$ in Latte parameters by setting $k = \left\lceil 1.17\sqrt{q/(2N)} \Big/ \sqrt{1/(2\ln 2)} \right\rceil$. This will also slightly increase σ_ℓ for all $\ell > 0$ since σ_ℓ is computed from σ_0 [ETS19]. Our revised Latte parameter sets are shown in Table 6.1. Compared to the original Latte parameters in Table 3.1, σ_ℓ is only at most 0.28% bigger for all $\ell \geq 0$ in our revised Latte parameter sets. In addition, our security analysis [ZMS+21] shows that the revised Latte parameters increase the decryption failure rate by at most 1 bit compared to the original Latte parameters in [ETS19].

Table 6.1: Revised Latte Parameters.

Set	**Security**	N	q	σ_ℓ		
				$\ell = 0$	$\ell = 1$	$\ell = 2$
Latte-1	128	1024	$2^{24} - 2^{14} + 1$	106.2	5513.3	-
Latte-2	256	2048	$2^{25} - 2^{12} + 1$	106.2	7900.2	-
Latte-3	80	1024	$2^{36} - 2^{20} + 1$	6777.6	351968.4	22559988.0
Latte-4	160	2048	$2^{38} - 2^{26} + 1$	9583.7	713170.8	65489528.1

Let $\mathbf{S}_\ell = \mathbf{L} \cdot \tilde{\mathbf{S}}_\ell$ be the Gram-Schmidt Orthogonal (GSO) decomposition of the delegated basis $\mathbf{S}_\ell \in \mathfrak{R}^{(\ell+2)\times(\ell+2)}$, where $\mathbf{L}$ is unit lower triangular and rows $\tilde{\mathbf{s}}_i$ of $\tilde{\mathbf{S}}_\ell$ are pairwise orthogonal. We find that the Euclidean norm of the last GSO vector $\tilde{\mathbf{s}}_{\ell+1}$ is very small (less than 1 in our experiment for Latte-3 and 4) compared to $\tilde{\mathbf{s}}_0, \ldots, \tilde{\mathbf{s}}_\ell$. This is because rows $\mathbf{s}_0, \ldots, \mathbf{s}_\ell$ of $\mathbf{S}_\ell$ are sampled with a large σ_ℓ but $\det(\mathbf{S}_\ell \cdot \mathbf{S}_\ell^*) = \prod_{i=0}^{\ell+1} \langle \tilde{\mathbf{s}}_i, \tilde{\mathbf{s}}_i \rangle$ is constant and always equal to q^2 [CPS+20]. The experiment results in Fig.3 of [CKKS19] also verified that $\|\tilde{\mathbf{s}}_{\ell+1}\|$ decreases significantly by increasing $\|\mathbf{s}_0\|$ for $\mathbf{S}_\ell \in \mathfrak{R}^{3\times 3}$. In this case, the ratio between the maximal and minimal standard deviation σ' used by the integer discrete Gaussian sampling subroutine in ffSampling (Line 4, 5 in Algorithm 2.3) is very large and the isochronous sampler [HPRR20] used by the constant-time Falcon implementation [PRR19] will be inefficient for our scheme, since the rejection rate of [HPRR20] is proportional to $(\max(\sigma')/\min(\sigma'))$. In order to sample with σ' in a broad range, we

employ a variant [SZJ+21] of the COSAC sampler in Chapter 5 instead, which is scalable to large σ' without sacrificing the efficiency.

To accelerate the Latte Encrypt and Decrypt speed, we change the distribution of ephemeral private keys $\mathbf{e}, \mathbf{e}_1, \ldots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b$ from $\mathscr{D}_{\sigma_e}$ (Line 4 in Algorithm 3.11; Line 4 in Algorithm 3.12) to a binomial distribution with center 0 and small standard deviation $\sigma_e = 2.0$ (Line 4 in Algorithm 6.5; Line 4 in Algorithm 6.6). Sampling from a binomial distribution is much faster than sampling from $\mathscr{D}_{\sigma_e}$ and the impact on security is negligible in the encryption [ADPS15]. Our optimised Latte Encrypt and Decrypt algorithms are shown in Algorithm 6.5 and Algorithm 6.6, respectively.

Algorithm 6.5 Optimised Latte Encrypt algorithm (at level ℓ).

Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \mathrm{KDF}, \mathsf{ID}_\ell, \mu \in \{0,1\}^{256}$.
Output: $Z \in \{0,1\}^{256}, \mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathfrak{R}_q$.

1: **function** Encrypt
2: $seed \hookleftarrow \{0,1\}^{256}$.
3: $Z \leftarrow \mu \oplus \mathrm{KDF}(seed)$.
4: Sample $\mathbf{e}, \mathbf{e}_1, \ldots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b$ from a binomial distribution with center 0 and standard deviation σ_e using the seed $\mathrm{KDF}(seed|Z)$.
5: **for** $i = 1$ **to** ℓ **do**
6: $\mathbf{C}_i \leftarrow \mathbf{A}_i \cdot \mathbf{e} + \mathbf{e}_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1|\ldots|\mathsf{ID}_i)$ in NTT domain.
7: **end for**
8: $\mathbf{C}_h \leftarrow \mathbf{h} \cdot \mathbf{e} + \mathbf{e}_h$.
9: $\mathbf{m} \leftarrow \mathrm{Encode}(seed)$.
10: $\mathbf{C}_b \leftarrow \mathbf{b} \cdot \mathbf{e} + \mathbf{e}_b + \mathbf{m}$.
11: **return** $Z \in \{0,1\}^{256}, \mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathfrak{R}_q$ in NTT domain.
12: **end function**

Algorithm 6.6 Optimised Latte Decrypt algorithm (at level ℓ).

Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \mathrm{KDF}, \mathsf{ID}_\ell, Z, (\mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b), (\mathbf{t}_0, \ldots, \mathbf{t}_{\ell+1})$.
Output: μ'.

1: **function** Decrypt
2: $\mathbf{V} \leftarrow \mathbf{C}_b - \mathbf{C}_h \cdot \mathbf{t}_1 - \mathbf{C}_1 \cdot \mathbf{t}_2 - \cdots - \mathbf{C}_\ell \cdot \mathbf{t}_{\ell+1}$.
3: $seed' \leftarrow \mathrm{Decode}(\mathbf{V})$.
4: Sample $\mathbf{e}', \mathbf{e}'_1, \ldots, \mathbf{e}'_\ell, \mathbf{e}'_h, \mathbf{e}'_b$ from a binomial distribution with center 0 and standard deviation σ_e using the seed $\mathrm{KDF}(seed'|Z)$.
5: **for** $i = 1$ **to** ℓ **do**
6: $\mathbf{C}'_i \leftarrow \mathbf{A}_i \cdot \mathbf{e}' + \mathbf{e}'_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1|\ldots|\mathsf{ID}_i)$ in NTT domain.
7: **end for**
8: $\mathbf{C}'_h \leftarrow \mathbf{h} \cdot \mathbf{e}' + \mathbf{e}'_h$.
9: $\mathbf{m}' \leftarrow \mathrm{Encode}(seed')$.
10: $\mathbf{C}'_b \leftarrow \mathbf{b} \cdot \mathbf{e}' + \mathbf{e}'_b + \mathbf{m}'$.
11: Check $(\mathbf{C}'_1, \ldots, \mathbf{C}'_\ell, \mathbf{C}'_h, \mathbf{C}'_b)$ agrees with $(\mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b)$, else return $\perp$.
12: **return** $\mu' = Z \oplus \mathrm{KDF}(seed')$.
13: **end function**

Due to the changes of parameters and distributions, our optimised Latte implementation will produce different Known Answer Test (KAT) responses compared to the original Latte scheme [ETS19]. Therefore, our optimised Latte scheme is incompatible (i.e. not a drop-in replacement) with the original Latte scheme and keys/ciphertexts of both schemes should not be mixed.

6.2 Optimised ffLDL Algorithm

The original generic FFT sampling procedure from Falcon [PFH+17] is discussed in Section 2.2.5, which includes the ffLDL algorithm for the Fast Fourier $\mathbf{LDL}^*$ decomposition (see Algorithm 2.1) and the ffSampling algorithm to sample from lattice discrete Gaussian distributions (see Algorithm 2.3). However, for the (Mod)NTRU basis $\mathbf{S}_\ell$ in Latte, we observe the following theorem, which can be adapted to accelerate the computation of the ffLDL algorithm:

Theorem 16. *In the ffLDL tree of the Gram matrix* $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^* \in (\mathbb{C}[x]/\langle x^N+1\rangle)^{d\times d}$ *in the FFT domain and* $\mathbf{S}_\ell$ *is a (Mod)NTRU basis, we have:*

1. $\forall i \in \{0,\dots,d-1\} : \mathbf{D}_{i,i} \in \mathbb{R}^n$ *for some* $n = 2^k \leq N$ *in every node of the tree.*
2. $\forall j \in \{0,\dots,N-1\} : \prod_{i=0}^{d-1}(\mathbf{D}_{i,i})_j = q^2$ *in the root of the tree.*
3. $\forall j \in \{0,\dots,n-1\} : (\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j = \mathbf{D}'_{2j}\mathbf{D}'_{2j+1}$ *for some* $n = 2^k \leq N/2$ *in every non-root node of the tree, where* $\mathbf{D}' \in \{\mathbf{D}_{i,i}\}_{i=0}^{d-1}$ *is from its parent.*
4. $\forall i \in \{0,\dots,d-1\}, j \in \{0,\dots,n-1\} : (\mathbf{D}_{i,i})_j \in \mathbb{R}^+$ *for some* $n = 2^k \leq N$ *in every node of the tree.*

Proof.

1. From Algorithm 2.1, we have $(\mathbf{D}_{0,0})_j = (\mathbf{G}_{0,0})_j$ and $(\mathbf{D}_{1,1})_j = (\mathbf{G}_{1,1})_j - |(\mathbf{L}_{1,0})_j|^2(\mathbf{G}_{0,0})_j$, $0 \leq j \leq n-1$, for some input matrix $\mathbf{G}$ in the FFT domain in every node of the tree. In addition, we have $(\mathbf{D}_{i,i})_j = (\mathbf{G}_{i,i})_j - \sum_{k<i}(|(\mathbf{L}_{i,k})_j|^2(\mathbf{D}_{k,k})_j)$ at the root when $d > 2$. Therefore, we have $\mathbf{D}_{i,i} \in \mathbb{R}^n$ assuming that $\mathbf{G}_{i,i} \in \mathbb{R}^n$ for all $i \in \{0,\dots,d-1\}$. To show that latter assumption is true, we observe that at the root we have the input $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ in the FFT domain, $\mathbf{G}_{i,i} \in \mathbb{R}^N$ for $0 \leq i \leq d-1$. Thus, $\mathbf{D}_{i,i} \in \mathbb{R}^N$ for $0 \leq i \leq d-1$ at the root. Assuming $\mathbf{D}_{i,i} \in \mathbb{R}^n$ for $0 \leq i \leq d-1$ at an non-leaf node, for its i-th child, we have the ffLDL input $\mathbf{G}'_{0,0} = \mathbf{G}'_{1,1} = \mathbf{d}_0$, where $(\mathbf{d}_0)_j = \frac{1}{2}[(\mathbf{D}_{i,i})_{2j} + (\mathbf{D}_{i,i})_{2j+1}] \in \mathbb{R}$ for $j \in \{0,\dots,n/2-1\}$. Thus, $\mathbf{D}'_{0,0}, \mathbf{D}'_{1,1} \in \mathbb{R}^{n/2}$ in this child and we can deduce the conclusion by induction.
2. Since by Definition 8, $\mathbf{L}$ is a lower triangular matrix with 1 on its diagonal and $\mathbf{D}$ is a diagonal matrix, we have $\det(\mathbf{D}) = \prod_{i=0}^{d-1}\mathbf{D}_{i,i} = \det(\mathbf{G})$. Because $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ at the

root and the determinant of a (Mod)NTRU basis $\mathbf{S}_\ell$ is q, we have $\prod_{i=0}^{d-1}(\mathbf{D}_{i,i})_j = q^2$ in the FFT domain at the root for $0 \leq j \leq N-1$.

3. For the i-th child of an non-leaf node, we have the ffLDL input $\mathbf{G}' = \begin{pmatrix} \mathbf{d}_0 & \mathbf{d}_1 \\ \mathbf{d}_1^* & \mathbf{d}_0 \end{pmatrix}$ for $\mathbf{d}_0, \mathbf{d}_1 \leftarrow \text{splitfft}(\mathbf{D}_{i,i})$, $0 \leq i \leq d-1$. By the definition of the $\mathbf{LDL}^*$ decomposition, for this child, we have $\mathbf{D}'_{0,0}\mathbf{D}'_{1,1} = \det(\mathbf{G}') = \mathbf{d}_0^2 - \mathbf{d}_1\mathbf{d}_1^*$. Thus, in the FFT domain, we have:

$$\begin{aligned}(\mathbf{D}'_{0,0})_j(\mathbf{D}'_{1,1})_j &= (\mathbf{d}_0)_j^2 - |(\mathbf{d}_1)_j|^2 \\ &= \left(\frac{1}{2}[(\mathbf{D}_{i,i})_{2j} + (\mathbf{D}_{i,i})_{2j+1}]\right)^2 \\ &\quad - \left|\frac{1}{2}[(\mathbf{D}_{i,i})_{2j} - (\mathbf{D}_{i,i})_{2j+1}]\omega^{-\text{bitrev}(n/2+j)}\right|^2,\end{aligned}$$

for $0 \leq j \leq n/2-1$. Since $(\mathbf{D}_{i,i})_{2j}, (\mathbf{D}_{i,i})_{2j+1} \in \mathbb{R}$ and $|\omega| = 1$, we get $(\mathbf{D}'_{0,0})_j(\mathbf{D}'_{1,1})_j = (\mathbf{D}_{i,i})_{2j}(\mathbf{D}_{i,i})_{2j+1}$.

4. The ffLDL algorithm computes the $\mathbf{LDL}^*$ decomposition in the FFT domain. Let $\mathbf{S}_\ell = \mathbf{L} \cdot \tilde{\mathbf{S}}_\ell$ be the GSO decomposition of $\mathbf{S}_\ell \in \mathfrak{R}^{d \times d}$ where rows of $\tilde{\mathbf{S}}_\ell$ are pairwise orthogonal. For the input $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ at the root, we have $\mathbf{G} = \mathbf{LDL}^*$ where $\mathbf{D} = \tilde{\mathbf{S}}_\ell\tilde{\mathbf{S}}_\ell^*$ [DP16]. Thus, in the FFT domain, $\mathbf{D}_{i,i} \in (\mathbb{R}^+)^N$ at the root. Assuming $\mathbf{D}_{i,i} \in (\mathbb{R}^+)^n$ for some $i \in \{0, \ldots, d-1\}$ at an non-leaf node, for the i-th child of this node, we have $(\mathbf{D}'_{0,0})_j(\mathbf{D}'_{1,1})_j = (\mathbf{D}_{i,i})_{2j}(\mathbf{D}_{i,i})_{2j+1} \in \mathbb{R}^+$ for $0 \leq j \leq n/2-1$. Because $(\mathbf{D}'_{0,0})_j = (\mathbf{d}_0)_j = \frac{1}{2}[(\mathbf{D}_{i,i})_{2j} + (\mathbf{D}_{i,i})_{2j+1}] \in \mathbb{R}^+$ due to the ffLDL input $\mathbf{G}' = \begin{pmatrix} \mathbf{d}_0 & \mathbf{d}_1 \\ \mathbf{d}_1^* & \mathbf{d}_0 \end{pmatrix}$ where $\mathbf{d}_0, \mathbf{d}_1 \leftarrow \text{splitfft}(\mathbf{D}_{i,i})$, we get $\mathbf{D}'_{0,0}, \mathbf{D}'_{1,1} \in (\mathbb{R}^+)^{n/2}$. Thus, we deduce the conclusion by induction.

□

We can utilise Theorem 16 when computing $\mathbf{D}$ in the ffLDL algorithm for the (Mod)NTRU basis $\mathbf{S}_\ell$ in Latte with $d \in \{2, 3\}$: $\mathbf{D}_{d-1,d-1}$ at the root can be computed by $(\mathbf{D}_{d-1,d-1})_j = \frac{q^2}{\prod_{i=0}^{d-2}(\mathbf{D}_{i,i})_j}$ for $0 \leq j \leq N-1$. For all the non-root nodes, we can directly compute $\mathbf{D}_{0,0}, \mathbf{D}_{1,1}$ by using $(\mathbf{D}_{0,0})_j = (\mathbf{G}_{0,0})_j$ and $(\mathbf{D}_{1,1})_j = \frac{\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{(\mathbf{D}_{0,0})_j}$, $0 \leq j \leq n-1$, for some $\mathbf{D}' \in \mathbb{R}^{2n}$, $\mathbf{G}_{0,0} = \mathbf{d}'_0 \in \mathbb{R}^n$ from its parent. Since we have $\forall i \in \{0, \ldots, d-1\} : \mathbf{D}_{i,i} \in \mathbb{R}^n$ in every node of the tree, the computation of $\mathbf{D}$ above can be done by solely using the real number arithmetic i.e. without complex number arithmetic. Because every complex number arithmetic computation contains multiple underlying floating-point (real number) arithmetic operations, by replacing complex number arithmetic with real number arithmetic when computing $\mathbf{D}$, we reduce the total amount of floating-point

arithmetic operations. Therefore, this optimisation technique will accelerate the run-time speed of the ffLDL algorithm (see Table 6.2 for the performance results).

Our optimised ffLDL algorithm for the (Mod)NTRU basis in Latte is shown in Algorithm 6.7. We also implement this algorithm and perform the benchmarks by using randomly generated (Mod)NTRU basis input $\mathbf{S}_\ell$ with the Latte parameter sets in Table 6.1. We employ the `mpfr` [FHL+07] library for multiprecision floating-point arithmetic, and we use the `mpc` [EGTZ18] library for multiprecision complex number arithmetic, respectively. The precision of floating-point and complex numbers in our implementation is $\lambda = 256$ bits. We obtain the benchmark results in terms of number of CPU cycles from a desktop machine with an Intel i7-7700K CPU at 4.2GHz, with both hyper-threading and TurboBoost disabled. We use the `gcc` 11.2.0 compiler with compiling options `-O3 -march=native` enabled. We also compare the run-time speed of our optimised ffLDL algorithm with a naive implementation[2] of the generic ffLDL algorithm in Algorithm 2.1. Both algorithms are implemented by using the same arithmetic libraries i.e. `mpfr` [FHL+07] and `mpc` [EGTZ18] with the same arithmetic precision $\lambda = 256$ bits in our benchmark.

The benchmark results are given in Table 6.2. From Table 6.2, our optimised ffLDL implementation for (Mod)NTRU basis in Latte achieves a 71.1%–73.4% speedup compared to the naive generic ffLDL implementation on average for all the Latte parameter sets, including both cases of $d = 2$ and $d = 3$.

Table 6.2: Comparison of the Average Number of CPU Cycles for ffLDL Algorithms in Latte.

	$d = 2$		$d = 3$	
Set	**Naive**	**Optimised**	**Naive**	**Optimised**
Latte-1	111456703	30225360	-	-
Latte-2	248906053	66146509	-	-
Latte-3	108951820	30369396	179996339	51995773
Latte-4	240682381	66045384	394954988	112447886

Since both the (Mod)Falcon digital signature schemes [PFH+17, CPS+20] and the Latte HIBE [ETS19] use similar (Mod)NTRU lattices [DLP14, CPS+20], our optimised ffLDL algorithm should also be applicable to (Mod)Falcon. However, we expect that the speedup under the Falcon settings [PFH+17] might be less significant than our Latte HIBE settings. This is because the speedup of our optimised ffLDL algorithm mainly comes from replacing complex number arithmetic with real number arithmetic, and the Falcon implementation [PFH+17] has lower complex number arithmetic overhead due to

[2]The naive implementation of the generic ffLDL algorithm in comparison was used in our old Latte implementation, which is available at https://gitlab.com/raykzhao/latte/-/tree/17228bd03d471ff94a6dab0da1ab53ba0e4b514b.

Algorithm 6.7 Optimised ffLDL algorithm for (Mod)NTRU basis in Latte.

Input: Gram matrix $\mathbf{G} \in (\mathbb{C}[x]/\langle x^n + 1\rangle)^{d\times d}$ in the FFT domain. $d \in \{2, 3\}$. $\mathbf{D'} \in (\mathbb{R}^+)^{2n}$.
Output: Tree T.

function ffLDL($\mathbf{G}, \mathbf{D'}$)
  if $n = 1$ **then**
    T.value $\leftarrow \mathbf{G}_{0,0}$.
  else
    $\mathbf{L} \leftarrow \mathbf{I}_d, \mathbf{D} \leftarrow \mathbf{0}_d$.
    for $j = 0$ **to** $n - 1$ **do**
      $(\mathbf{D}_{0,0})_j \leftarrow (\mathbf{G}_{0,0})_j$.
      $(\mathbf{L}_{1,0})_j \leftarrow \frac{(\mathbf{G}_{1,0})_j}{(\mathbf{D}_{0,0})_j}$.
      if $d = 2$ **then**
        if $n = N$ **then**
          $(\mathbf{D}_{1,1})_j \leftarrow \frac{q^2}{(\mathbf{D}_{0,0})_j}$.
        else
          $(\mathbf{D}_{1,1})_j \leftarrow \frac{\mathbf{D'}_{2j}\mathbf{D'}_{2j+1}}{(\mathbf{D}_{0,0})_j}$.
        end if
      else if $d = 3$ **then**
        $(\mathbf{D}_{1,1})_j \leftarrow (\mathbf{G}_{1,1})_j - \frac{|(\mathbf{G}_{1,0})_j|^2}{(\mathbf{D}_{0,0})_j}$.
        $(\mathbf{D}_{2,2})_j \leftarrow \frac{q^2}{(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j}$.
        $(\mathbf{L}_{2,0})_j \leftarrow \frac{(\mathbf{G}_{2,0})_j}{(\mathbf{D}_{0,0})_j}$.
        $(\mathbf{L}_{2,1})_j \leftarrow \frac{(\mathbf{G}_{2,1})_j - (\mathbf{G}_{2,0})_j(\mathbf{L}_{1,0})_j^*}{(\mathbf{D}_{1,1})_j}$.
      end if
    end for
    T.value $\leftarrow \mathbf{L}$.
    for $i = 0$ **to** $d - 1$ **do**
      $\mathbf{d}_0, \mathbf{d}_1 \leftarrow$ splitfft($\mathbf{D}_{i,i}$).
      $\mathbf{G'} = \begin{pmatrix} \mathbf{d}_0 & \mathbf{d}_1 \\ \mathbf{d}_1^* & \mathbf{d}_0 \end{pmatrix}$.
      T.child$_i \leftarrow$ ffLDL($\mathbf{G'}, \mathbf{D}_{i,i}$).
    end for
  end if
  return T.
end function

the lower floating-point arithmetic precision (double precision) compared to our current Latte HIBE settings (256-bit multiprecision).

6.3 ffLDL Error Analysis

The Rényi divergence argument [PFH+17, HPRR20] for the arithmetic precision requirement of the FFT sampling procedure from Falcon [PFH+17] needs the upper bound of the *relative* error δ_σ among the standard deviation σ and the upper bound of the *absolute* error Δ_c among the center c for the integer discrete Gaussian sampling subroutine in ffSampling, where σ is related to the leaf values of ffLDL, and c relies on the result of $\mathbf{L}$ in ffLDL. However, Falcon [PFH+17] only provided the heuristic error bounds based on experimental results of their parameter sets with very little technical discussion.

Since the leaf values of the ffLDL tree are computed from $\mathbf{D}$ of their parents in Algorithm 6.7, here we provide our provable theoretical error analysis of the optimised ffLDL algorithm in Algorithm 6.7 for the (Mod)NTRU basis in Latte, including the errors of both $\mathbf{D}$ (see Section 6.3.1) and $\mathbf{L}$ (see Section 6.3.2). In addition, we also provide the numerical values of our error bounds computed from the Latte parameter sets in Section 6.3.4.

6.3.1 Error Analysis of D in ffLDL

By Theorem 16, we have $\mathbf{D} \in (\mathbb{R}^n)^{d\times d}$ in every node of the ffLDL tree and the computation of $\mathbf{D}$ is done by solely using real number arithmetic in Algorithm 6.7. Let the *relative* error of the floating-point arithmetic be u i.e. $u = 2^{-p}$ for floating-point with precision p. In this section we derive the simplified upper bounds on the absolute errors in $\mathbf{D}$ values of the ffLDL tree in terms of a small number of quantities. We will bound these quantities in Section 6.3.4.

$\Delta_{\mathbf{D}_{0,0}}$ at the root: The absolute error bound $\Delta_{\mathbf{D}_{0,0}}$ at the root is $\Delta_{\mathbf{G}_{0,0}}$ for the input $\mathbf{G}$ in the FFT domain, since $(\mathbf{D}_{0,0})_j = (\mathbf{G}_{0,0})_j$, $0 \le j \le N-1$.

$\Delta_{\mathbf{D}_{1,1}}$ at the root: Since the computation of $\mathbf{D}_{1,1}$ is different between $d = 2$ and $d = 3$ at the root in Algorithm 6.7, here we analyse $\Delta_{\mathbf{D}_{1,1}}$ in these two cases:

When $d = 2$: Assume q can be exactly represented within the floating-point precision without rounding. By Theorem 16, since $(\mathbf{D}_{1,1})_j = q^2/(\mathbf{D}_{0,0})_j$ and $(\mathbf{D}_{0,0})_j \in \mathbb{R}^+$, $0 \le j \le N-1$, we have the following absolute error bound $\Delta_{\mathbf{D}_{1,1}}$. Here, we

assume that q^2 may contain errors i.e. q^2 may not be exactly represented within the floating-point precision. This is the case for q in the Latte-3 and 4 parameter sets in Table 6.1 when using e.g. double precision.

$$\begin{aligned}\Delta_{\mathbf{D}_{1,1}} &\leq \max_{j=0}^{N-1}\left[(1+u)\frac{(1+u)q^2}{(\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}}} - \frac{q^2}{(\mathbf{D}_{0,0})_j}\right] \\ &= \max_{j=0}^{N-1}\left[\frac{(u^2+2u)q^2}{(\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}}} + \frac{q^2\Delta_{\mathbf{D}_{0,0}}}{((\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}})(\mathbf{D}_{0,0})_j}\right] \\ &\leq \frac{(u^2+2u)q^2}{\min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}}} + \frac{q^2\Delta_{\mathbf{D}_{0,0}}}{\left(\min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}}\right)\cdot \min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j}.\end{aligned}$$

When $d = 3$: From Line 16 in Algorithm 6.7, we have $(\mathbf{D}_{1,1})_j = (\mathbf{G}_{1,1})_j - |(\mathbf{G}_{1,0})_j|^2/(\mathbf{D}_{0,0})_j$, $0 \leq j \leq N-1$. By Theorem 16 and $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ in the FFT domain, we have $(\mathbf{G}_{1,1})_j, (\mathbf{D}_{0,0})_j, (\mathbf{D}_{1,1})_j \in \mathbb{R}^+$. Thus, we have the following absolute error bound $\Delta_{\mathbf{D}_{1,1}}$:

$$\begin{aligned}\Delta_{\mathbf{D}_{1,1}} \leq \max_{j=0}^{N-1}\Bigg[&(1+u)\left((\mathbf{G}_{1,1})_j + \Delta_{\mathbf{G}_{1,1}} - (1-u)\frac{(1-u)(|(\mathbf{G}_{1,0})_j| - \Delta_{\mathbf{G}_{1,0}})^2}{(\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}}}\right) \\ &-\left((\mathbf{G}_{1,1})_j - \frac{|(\mathbf{G}_{1,0})_j|^2}{(\mathbf{D}_{0,0})_j}\right)\Bigg] \\ = \max_{j=0}^{N-1}\Bigg[&u\cdot(\mathbf{G}_{1,1})_j + (1+u)\Delta_{\mathbf{G}_{1,1}} \\ &+\frac{|(\mathbf{G}_{1,0})_j|^2 - (1+u)(1-u)^2(|(\mathbf{G}_{1,0})_j| - \Delta_{\mathbf{G}_{1,0}})^2}{(\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}}} \\ &+\frac{|(\mathbf{G}_{1,0})_j|^2\Delta_{\mathbf{D}_{0,0}}}{((\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}})(\mathbf{D}_{0,0})_j}\Bigg].\end{aligned}$$

Since $(\mathbf{D}_{1,1})_j = (\mathbf{G}_{1,1})_j - |(\mathbf{G}_{1,0})_j|^2/(\mathbf{D}_{0,0})_j \in \mathbb{R}^+$, we have $|(\mathbf{G}_{1,0})_j|^2/(\mathbf{D}_{0,0})_j \leq (\mathbf{G}_{1,1})_j$ and $|(\mathbf{G}_{1,0})_j| \leq \sqrt{(\mathbf{G}_{1,1})_j(\mathbf{D}_{0,0})_j}$ for $(\mathbf{G}_{1,1})_j, (\mathbf{D}_{0,0})_j \in \mathbb{R}^+$. Therefore,

$$\begin{aligned}\Delta_{\mathbf{D}_{1,1}} &\leq \max_{j=0}^{N-1}\Bigg[u\cdot(\mathbf{G}_{1,1})_j + (1+u)\Delta_{\mathbf{G}_{1,1}} + (1-(1+u)(1-u)^2)(\mathbf{G}_{1,1})_j \\ &\quad +\frac{2(1+u)(1-u)^2\Delta_{\mathbf{G}_{1,0}}\sqrt{(\mathbf{G}_{1,1})_j}}{\sqrt{(\mathbf{D}_{0,0})_j}} + \frac{\Delta_{\mathbf{D}_{0,0}}}{(\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}}}(\mathbf{G}_{1,1})_j\Bigg] \\ &\leq \left(2u + u^2 - u^3 + \frac{\Delta_{\mathbf{D}_{0,0}}}{\min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}}}\right)\cdot \max_{j=0}^{N-1}(\mathbf{G}_{1,1})_j + (1+u)\Delta_{\mathbf{G}_{1,1}} \\ &\quad +\frac{2(1+u)(1-u)^2\Delta_{\mathbf{G}_{1,0}}\sqrt{\max_{j=0}^{N-1}(\mathbf{G}_{1,1})_j}}{\sqrt{\min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j}}.\end{aligned}$$

$\Delta_{\mathbf{D}_{2,2}}$ at the root ($d = 3$): Assume q can be exactly represented within the floating-point precision without rounding. By Theorem 16, since $(\mathbf{D}_{2,2})_j = \frac{q^2}{(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j}$ and $(\mathbf{D}_{0,0})_j, (\mathbf{D}_{1,1})_j \in \mathbb{R}^+$, $0 \le j \le N-1$, we have the following absolute error bound $\Delta_{\mathbf{D}_{2,2}}$. Here, we also assume that q^2 may not be exactly represented within the floating-point precision and may contain errors.

$$\begin{aligned}\Delta_{\mathbf{D}_{2,2}} &\le \max_{j=0}^{N-1}\left[(1+u)\frac{(1+u)q^2}{(1-u)((\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}})((\mathbf{D}_{1,1})_j - \Delta_{\mathbf{D}_{1,1}})} - \frac{q^2}{(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j}\right] \\ &= \max_{j=0}^{N-1}\left[\frac{(u^2+3u)q^2}{(1-u)((\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}})((\mathbf{D}_{1,1})_j - \Delta_{\mathbf{D}_{1,1}})}\right. \\ &\left. + \frac{q^2((\mathbf{D}_{0,0})_j\Delta_{\mathbf{D}_{1,1}} + (\mathbf{D}_{1,1})_j\Delta_{\mathbf{D}_{0,0}} - \Delta_{\mathbf{D}_{0,0}}\Delta_{\mathbf{D}_{1,1}})}{((\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}})((\mathbf{D}_{1,1})_j - \Delta_{\mathbf{D}_{1,1}})(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j}\right]\end{aligned}$$

For Latte parameter sets in Table 6.1, we have the empirical observation $(\mathbf{D}_{2,2})_j \in [0,1]$, $0 \le j \le N-1$, at the root when $d = 3$ in our experiment. Since $(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j(\mathbf{D}_{2,2})_j = q^2$ at the root from Theorem 16, we have $(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j \ge q^2$. Then,

$$\begin{aligned}&((\mathbf{D}_{0,0})_j - \Delta_{\mathbf{D}_{0,0}})((\mathbf{D}_{1,1})_j - \Delta_{\mathbf{D}_{1,1}}) \\ &= (\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j - (\mathbf{D}_{0,0})_j\Delta_{\mathbf{D}_{1,1}} - (\mathbf{D}_{1,1})_j\Delta_{\mathbf{D}_{0,0}} + \Delta_{\mathbf{D}_{0,0}}\Delta_{\mathbf{D}_{1,1}} \\ &\ge q^2 - (\mathbf{D}_{0,0})_j\Delta_{\mathbf{D}_{1,1}} - (\mathbf{D}_{1,1})_j\Delta_{\mathbf{D}_{0,0}} + \Delta_{\mathbf{D}_{0,0}}\Delta_{\mathbf{D}_{1,1}}.\end{aligned}$$

Let $\mathbf{x}$ be the common terms in the numerator and denominator:

$$\mathbf{x}_j = (\mathbf{D}_{0,0})_j\Delta_{\mathbf{D}_{1,1}} + (\mathbf{D}_{1,1})_j\Delta_{\mathbf{D}_{0,0}} - \Delta_{\mathbf{D}_{0,0}}\Delta_{\mathbf{D}_{1,1}},$$

for $0 \le j \le N-1$. Assuming $q^2 > \mathbf{x}_j$, we have:

$$\begin{aligned}\Delta_{\mathbf{D}_{2,2}} &\le \max_{j=0}^{N-1}\frac{(u^2+3u)q^2 + (1-u)\mathbf{x}_j}{(1-u)(q^2 - \mathbf{x}_j)} \\ &\le \frac{(u^2+3u)q^2 + (1-u)\cdot\max_{j=0}^{N-1}\mathbf{x}_j}{(1-u)\left(q^2 - \max_{j=0}^{N-1}\mathbf{x}_j\right)}.\end{aligned}$$

where

$$\mathbf{x}_j \le \Delta_{\mathbf{D}_{1,1}}\cdot\max_{j=0}^{N-1}(\mathbf{D}_{0,0})_j + \Delta_{\mathbf{D}_{0,0}}\cdot\max_{j=0}^{N-1}(\mathbf{D}_{1,1})_j - \Delta_{\mathbf{D}_{0,0}}\Delta_{\mathbf{D}_{1,1}}.$$

$\Delta_{\mathbf{D}_{0,0}}$ at non-root nodes: For a non-root node, from Line 7, 24 in Algorithm 6.7 and by Definition 9, we have $(\mathbf{D}_{0,0})_j = \frac{1}{2}(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1})$, $0 \le j \le n-1$, for some $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$

from its parent. Thus, we have the following absolute error bound $\Delta_{\mathbf{D}_{0,0}}$:

$$\begin{aligned}\Delta_{\mathbf{D}_{0,0}} &\leq \max_{j=0}^{n-1}\left[\frac{1}{2}(1+u)(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}+2\Delta_{\mathbf{D}'})-\frac{1}{2}(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})\right]\\ &= \max_{j=0}^{n-1}\left[\frac{1}{2}u(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})+(1+u)\Delta_{\mathbf{D}'}\right].\end{aligned}$$

$\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}$ gets the maximal value when both $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1}$ have the largest value of $\mathbf{D}'$. Thus,

$$\Delta_{\mathbf{D}_{0,0}} \leq u \cdot \max_{j=0}^{2n-1}\mathbf{D}'_j + (1+u)\Delta_{\mathbf{D}'}.$$

Note that T.value at leaves in Algorithm 6.7 is equal to $\mathbf{D}_{0,0}$. Therefore, the absolute error of T.value at leaves is $\Delta_{\mathbf{D}_{0,0}}$.

$\Delta_{\mathbf{D}_{1,1}}$ at non-root, non-leaf nodes: For a non-root, non-leaf node, from Line 13, 24 in Algorithm 6.7 and by Definition 9, we have $(\mathbf{D}_{1,1})_j = \mathbf{D}'_{2j}\mathbf{D}'_{2j+1}/(\mathbf{D}_{0,0})_j = \frac{\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{1/2\cdot(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}$, $0 \leq j \leq n-1$, for some $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$ from its parent. For $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$, $\frac{\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{1/2\cdot(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}$ gets the maximal value when both $\mathbf{D}'_{2j}$ and $\mathbf{D}'_{2j+1}$ get their maximal values. Thus, we have the following absolute error bound $\Delta_{\mathbf{D}_{1,1}}$:

$$\begin{aligned}\Delta_{\mathbf{D}_{1,1}} &\leq \max_{j=0}^{n-1}\left[(1+u)\frac{(1+u)(\mathbf{D}'_{2j}+\Delta_{\mathbf{D}'})(\mathbf{D}'_{2j+1}+\Delta_{\mathbf{D}'})}{\frac{1}{2}(1-u)(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}+2\Delta_{\mathbf{D}'})}-\frac{\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{\frac{1}{2}(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}\right]\\ &\leq \max_{j=0}^{n-1}\left[\frac{2(u^2+3u)\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{(1-u)(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}\right.\\ &\left.+\frac{2\Delta_{\mathbf{D}'}(1+u)^2((\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})^2+\Delta_{\mathbf{D}'}(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}))}{(1-u)(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}+2\Delta_{\mathbf{D}'})(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}\right].\end{aligned}$$

Since $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$, we have:

$$(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})^2+\Delta_{\mathbf{D}'}(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}) \leq (\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}+2\Delta_{\mathbf{D}'})(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1}).$$

Thus, we get:

$$\Delta_{\mathbf{D}_{1,1}} \leq \frac{(u^2+3u)\cdot\max_{j=0}^{2n-1}\mathbf{D}'_j + 2(1+u)^2\Delta_{\mathbf{D}'}}{1-u}.$$

From Algorithm 2.2, the standard deviation σ of the integer discrete Gaussian in ffSampling is $\sigma_\ell/\sqrt{\text{leaf.value}}$ for some σ_ℓ. By combining all the error analysis above, since

T.value at leaves is equal to $\mathbf{D}_{0,0}$, we get the following *relative* error bound δ_σ:

$$\begin{aligned}\delta_\sigma &\leq \max_{\text{all leaves}} \left[\frac{(1+u)\frac{(1+u)\sigma_\ell}{(1-u)\sqrt{\mathbf{D}_{0,0}-\Delta_{\mathbf{D}_{0,0}}}}}{\frac{\sigma_\ell}{\sqrt{\mathbf{D}_{0,0}}}} - 1 \right] \\ &= \frac{(1+u)^2}{1-u}\sqrt{\max_{\text{all leaves}} \frac{\mathbf{D}_{0,0}}{\mathbf{D}_{0,0}-\Delta_{\mathbf{D}_{0,0}}}} - 1.\end{aligned}$$

6.3.2 Error Analysis of L in ffLDL

Here, we analyse the absolute error of $\mathbf{L}$ in our optimised ffLDL algorithm in Algorithm 6.7. Similar to Section 6.3.1, in this section we derive the simplified upper bounds on the absolute errors in $\mathbf{L}$ values of the ffLDL tree in terms of a small number of quantities. We will bound these quantities in Section 6.3.4.

$\Delta_{\mathbf{L}_{1,0}}$ at the root: We have $(\mathbf{L}_{1,0})_j = (\mathbf{G}_{1,0})_j/(\mathbf{D}_{0,0})_j$, $0 \leq j \leq N-1$, at the root for the input $\mathbf{G} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ in the FFT domain. By Theorem 16, since $(\mathbf{D}_{0,0})_j \in \mathbb{R}^+$, we have the upper bound of $|(\mathbf{L}_{1,0})_j|$:

$$|(\mathbf{L}_{1,0})_j| \leq \frac{\max_{k=0}^{N-1}|(\mathbf{G}_{1,0})_k|}{\min_{k=0}^{N-1}(\mathbf{D}_{0,0})_k},$$

and the absolute error bound:

$$\Delta_{\mathbf{L}_{1,0}} \leq \Delta_{/\mathbb{R}}\left(\max_{j=0}^{N-1}|(\mathbf{G}_{1,0})_j|, \min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j, \Delta_{\mathbf{G}_{1,0}}, \Delta_{\mathbf{D}_{0,0}}\right).$$

$\Delta_{\mathbf{L}_{2,0}}$ at the root ($d = 3$): Similar to $\mathbf{L}_{1,0}$, since $(\mathbf{L}_{2,0})_j = (\mathbf{G}_{2,0})_j/(\mathbf{D}_{0,0})_j$, $0 \leq j \leq N-1$, at the root for the input $\mathbf{G}$ in the FFT domain, we have the upper bound of $|(\mathbf{L}_{2,0})_j|$:

$$|(\mathbf{L}_{2,0})_j| \leq \frac{\max_{k=0}^{N-1}|(\mathbf{G}_{2,0})_k|}{\min_{k=0}^{N-1}(\mathbf{D}_{0,0})_k},$$

and the absolute error bound:

$$\Delta_{\mathbf{L}_{2,0}} \leq \Delta_{/\mathbb{R}}\left(\max_{j=0}^{N-1}|(\mathbf{G}_{2,0})_j|, \min_{j=0}^{N-1}(\mathbf{D}_{0,0})_j, \Delta_{\mathbf{G}_{2,0}}, \Delta_{\mathbf{D}_{0,0}}\right).$$

$\Delta_{\mathbf{L}_{2,1}}$ at the root ($d = 3$): From Line 19 in Algorithm 6.7, we have $(\mathbf{L}_{2,1})_j = \frac{(\mathbf{G}_{2,1})_j - (\mathbf{G}_{2,0})_j(\mathbf{L}_{1,0})_j^*}{(\mathbf{D}_{1,1})_j}$, $0 \leq j \leq N-1$, at the root for the input $\mathbf{G}$ in the FFT domain. Let $\mathbf{x}_j = (\mathbf{G}_{2,1})_j - (\mathbf{G}_{2,0})_j(\mathbf{L}_{1,0})_j^*$ be the numerator of $(\mathbf{L}_{2,1})_j$. We have the upper bound of

$|\mathbf{x}_j|$:

$$|\mathbf{x}_j| \leq \max_{k=0}^{N-1} |(\mathbf{G}_{2,1})_k| + \left(\max_{k=0}^{N-1} |(\mathbf{G}_{2,0})_k|\right)\left(\max_{k=0}^{N-1} |(\mathbf{L}_{1,0})_k|\right),$$

and the absolute error bound:

$$\Delta_{\mathbf{x}} \leq \Delta_{\pm}\left[\max_{j=0}^{N-1} |(\mathbf{G}_{2,1})_j|, \left(\max_{j=0}^{N-1} |(\mathbf{G}_{2,0})_j|\right)\left(\max_{j=0}^{N-1} |(\mathbf{L}_{1,0})_j|\right), \Delta_{\mathbf{G}_{2,1}},\right.$$
$$\left.\Delta_{\times}\left(\max_{j=0}^{N-1} |(\mathbf{G}_{2,0})_j|, \max_{j=0}^{N-1} |(\mathbf{L}_{1,0})_j|, \Delta_{\mathbf{G}_{2,0}}, \Delta_{\mathbf{L}_{1,0}}\right)\right].$$

By Theorem 16, since $(\mathbf{D}_{1,1})_j \in \mathbb{R}^+$, for $(\mathbf{L}_{2,1})_j = \mathbf{x}_j/(\mathbf{D}_{1,1})_j$, we have the upper bound of $|(\mathbf{L}_{2,1})_j|$:

$$|(\mathbf{L}_{2,1})_j| \leq \frac{\max_{k=0}^{N-1} |\mathbf{x}_k|}{\min_{k=0}^{N-1} (\mathbf{D}_{1,1})_k},$$

and the absolute error bound:

$$\Delta_{\mathbf{L}_{2,1}} \leq \Delta_{/\mathbb{R}}\left(\max_{j=0}^{N-1} |\mathbf{x}_j|, \min_{j=0}^{N-1} (\mathbf{D}_{1,1})_j, \Delta_{\mathbf{x}}, \Delta_{\mathbf{D}_{1,1}}\right).$$

$\Delta_{\mathbf{L}_{1,0}}$ at non-root, non-leaf nodes: From Line 8, 24 in Algorithm 6.7 and by Definition 9, at a non-root, non-leaf node, we have:

$$(\mathbf{L}_{1,0})_j = \frac{(\mathbf{G}'_{1,0})_j}{(\mathbf{D}_{0,0})_j} = \frac{[(\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1})\omega^{-\text{bitrev}(n/2+j)}]^*}{\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}},$$

$0 \leq j \leq n-1$, for some $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$ from its parent, where ω is the $2N$-th complex root of unity. Since $|\omega| = 1$, for $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$, we have:

$$|(\mathbf{L}_{1,0})_j| = \frac{|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}|}{\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}} \leq \frac{\max_{k=0}^{2n-1} \mathbf{D}'_k - \min_{k=0}^{2n-1} \mathbf{D}'_k}{\max_{k=0}^{2n-1} \mathbf{D}'_k + \min_{k=0}^{2n-1} \mathbf{D}'_k}.$$

The inequality above is because $\frac{|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}|}{\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}}$ has the maximal value when one of $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$ gets the largest value $\max_{k=0}^{2n-1} \mathbf{D}'_k$ and the other gets the smallest value $\min_{k=0}^{2n-1} \mathbf{D}'_k$.

For the absolute error $\Delta_{\mathbf{L}_{1,0}}$, we have:

$$\Delta_{\mathbf{L}_{1,0}} \leq \max_{j=0}^{n-1} \Delta_{/\mathbb{R}}[|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| \cdot |\omega^{-\text{bitrev}(n/2+j)}|, \mathbf{D}'_{2j} + \mathbf{D}'_{2j+1},$$
$$\Delta_{\times\mathbb{R}}(|\omega^{-\text{bitrev}(n/2+j)}|, |\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}|, \Delta_{\omega}, \Delta_{(\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1})}), \Delta_{(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1})}],$$

where the absolute error of complex root of unity $\Delta_\omega \leq u/\sqrt{2}$ [BJM+20]. For the absolute error of $\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}$, since $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$, we have:

$$\begin{aligned}\Delta_{(\mathbf{D}'_{2j}-\mathbf{D}'_{2j+1})} &\leq (1+u)(|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| + 2\Delta_{\mathbf{D}'}) - |\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| \\ &= u|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| + 2(1+u)\Delta_{\mathbf{D}'}.\end{aligned}$$

Similarly, for the absolute error of $\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}$, we have:

$$\begin{aligned}\Delta_{(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})} &\leq (1+u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1} + 2\Delta_{\mathbf{D}'}) - (\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) \\ &= u(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) + 2(1+u)\Delta_{\mathbf{D}'}.\end{aligned}$$

Therefore, we have:

$$\begin{aligned}\Delta_{\mathbf{L}_{1,0}} \leq & \frac{C_1|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| + C_2\Delta_{\mathbf{D}'}}{(1-u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) - 2(1+u)\Delta_{\mathbf{D}'}} \\ & + \frac{2(1+u)\Delta_{\mathbf{D}'}|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}|}{[(1-u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) - 2(1+u)\Delta_{\mathbf{D}'}](\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1})},\end{aligned}$$

where

$$\begin{aligned}C_1 &= u\left[\frac{\sqrt{2}}{2}u^3 + \left(\frac{3\sqrt{2}}{2}+1\right)u^2 + \left(\frac{3\sqrt{2}}{2}+3\right)u + \frac{\sqrt{2}}{2} + 4\right], \\ C_2 &= \sqrt{2}u^4 + (3\sqrt{2}+2)u^3 + (3\sqrt{2}+6)u^2 + (\sqrt{2}+6)u + 2.\end{aligned}$$

Since $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} \in \mathbb{R}^+$, we have $|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| \leq \mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}$. Then, for the second term in the inequality of $\Delta_{\mathbf{L}_{1,0}}$ above, we have:

$$\begin{aligned}&\frac{2(1+u)\Delta_{\mathbf{D}'}|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}|}{[(1-u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) - 2(1+u)\Delta_{\mathbf{D}'}](\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1})} \\ &\leq \frac{2(1+u)\Delta_{\mathbf{D}'}}{(1-u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) - 2(1+u)\Delta_{\mathbf{D}'}}.\end{aligned}$$

Thus, we have:

$$\begin{aligned}\Delta_{\mathbf{L}_{1,0}} &\leq \frac{C_1|\mathbf{D}'_{2j} - \mathbf{D}'_{2j+1}| + C'_2\Delta_{\mathbf{D}'}}{(1-u)(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1}) - 2(1+u)\Delta_{\mathbf{D}'}} \\ &\leq \frac{C_1\left(\max_{k=0}^{2n-1}\mathbf{D}'_k - \min_{k=0}^{2n-1}\mathbf{D}'_k\right) + C'_2\Delta_{\mathbf{D}'}}{2(1-u)\cdot\min_{k=0}^{2n-1}\mathbf{D}'_k - 2(1+u)\Delta_{\mathbf{D}'}},\end{aligned}$$

where

$$C'_2 = \sqrt{2}u^4 + (3\sqrt{2}+2)u^3 + (3\sqrt{2}+6)u^2 + (\sqrt{2}+8)u + 4.$$

6.3.3 ffLDL Error Computation Algorithm

Combining the error analysis of $\mathbf{D}$ in Section 6.3.1 and the error analysis of $\mathbf{L}$ in Section 6.3.2, we can compute $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ recursively for every node in the ffLDL tree. Since both $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ at non-root nodes need the bounds $\max_{j=0}^{2n-1} \mathbf{D}'_j$ and $\min_{j=0}^{2n-1} \mathbf{D}'_j$ for some $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$ from its parent, here we provide the following lemma to analyse the maximal and minimal values of $\mathbf{D}'$:

Lemma 4. *For every non-root, non-leaf node in an ffLDL tree, we have:*

$$\min_{k=0}^{2n-1} \mathbf{D}'_k \leq (\mathbf{D}_{i,i})_j \leq \max_{k=0}^{2n-1} \mathbf{D}'_k,$$

$i \in \{0, 1\}$, $0 \leq j \leq n-1$, *for some* $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$ *from its parent.*

Proof. From Theorem 16, for a non-root, non-leaf node, since $(\mathbf{D}_{0,0})_j = \frac{1}{2}(\mathbf{D}'_{2j} + \mathbf{D}'_{2j+1})$, $0 \leq j \leq n-1$, for some $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$, $(\mathbf{D}_{0,0})_j$ gets the minimal value $\min_{k=0}^{2n-1} \mathbf{D}'_k$ when both $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} = \min_{k=0}^{2n-1} \mathbf{D}'_k$. Similarly, $(\mathbf{D}_{0,0})_j$ gets the maximal value $\max_{k=0}^{2n-1} \mathbf{D}'_k$ when both $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} = \max_{k=0}^{2n-1} \mathbf{D}'_k$. For $(\mathbf{D}_{1,1})_j = \mathbf{D}'_{2j}\mathbf{D}'_{2j+1}/(\mathbf{D}_{0,0})_j = \frac{\mathbf{D}'_{2j}\mathbf{D}'_{2j+1}}{1/2\cdot(\mathbf{D}'_{2j}+\mathbf{D}'_{2j+1})}$, it gets the minimal value $\min_{k=0}^{2n-1} \mathbf{D}'_k$ when both $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} = \min_{k=0}^{2n-1} \mathbf{D}'_k$ and $(\mathbf{D}_{1,1})_j$ gets the maximal value $\max_{k=0}^{2n-1} \mathbf{D}'_k$ when both $\mathbf{D}'_{2j}, \mathbf{D}'_{2j+1} = \max_{k=0}^{2n-1} \mathbf{D}'_k$ for $\mathbf{D}' \in (\mathbb{R}^+)^{2n}$. □

From Lemma 4, if the ancestor of a non-root, non-leaf node is the m-th child of the root, $0 \leq m \leq d-1$, then $(\mathbf{D}_{i,i})_j$ of this node has the minimal value $\min_{k=0}^{N-1}(\mathbf{D}'_{m,m})_k$ and the maximal value $\max_{k=0}^{N-1}(\mathbf{D}'_{m,m})_k$, $i \in \{0, 1\}$, $0 \leq j \leq n-1$, for $\mathbf{D}'_{m,m}$ from the root, respectively. Therefore, we can first compute $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ of the root and then use the recursive algorithm in Algorithm 6.8 to compute $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ for every non-root node by using the results from the root as the input.

6.3.4 Practical Implication

In order to compute $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ at the root for the input $\mathbf{G} = \mathbf{S}_\ell \mathbf{S}_\ell \in (\mathbb{C}[x]/\langle x^N + 1\rangle)^{d\times d}$ in the FFT domain, where $\mathbf{S}_\ell$ is a (Mod)NTRU basis in Latte, from the analysis in Section 6.3.1 and Section 6.3.2, we need the following bounds of elements from the input $\mathbf{G}$:

When $d = 2$**:** $\Delta_{\mathbf{G}_{0,0}}$, $\max_{j=0}^{N-1} |(\mathbf{G}_{1,0})_j|$, $\Delta_{\mathbf{G}_{1,0}}$.

When $d = 3$**:** $\Delta_{\mathbf{G}_{0,0}}$, $\max_{j=0}^{N-1}(\mathbf{G}_{1,1})_j$, $\Delta_{\mathbf{G}_{1,1}}$, $\max_{j=0}^{N-1} |(\mathbf{G}_{1,0})_j|$, $\Delta_{\mathbf{G}_{1,0}}$, $\max_{j=0}^{N-1} |(\mathbf{G}_{2,0})_j|$, $\Delta_{\mathbf{G}_{2,0}}$, $\max_{j=0}^{N-1} |(\mathbf{G}_{2,1})_j|$, $\Delta_{\mathbf{G}_{2,1}}$.

Algorithm 6.8 Computation of $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ for non-root nodes in the ffLDL tree.

Input: $m_1 = \max_{k=0}^{2n-1}(\mathbf{D'})_k, m_2 = \min_{k=0}^{2n-1}(\mathbf{D'})_k, \Delta_{\mathbf{D'}}$ for some $\mathbf{D'}$ from the parent. Ring dimension n. Relative error u of floating-point arithmetic.

Output: $\Delta_{\mathbf{D}_{0,0}}, \Delta_{\mathbf{D}_{1,1}}, \Delta_{\mathbf{L}_{1,0}}$.

1: **function** ffLDLErr($m_1, m_2, \Delta_{\mathbf{D'}}, n$)
2: $\quad\Delta_{\mathbf{D}_{0,0}} \leftarrow um_1 + (1+u)\Delta_{\mathbf{D'}}$.
3: $\quad$**if** $n = 1$ **then**
4: $\quad\quad\delta_\sigma \leftarrow \frac{(1+u)^2}{1-u}\sqrt{\frac{m_2}{m_2-\Delta_{\mathbf{D}_{0,0}}}} - 1$.
5: $\quad$**else**
6: $\quad\quad\Delta_{\mathbf{D}_{1,1}} \leftarrow \frac{(u^2+3u)m_1+2(1+u)^2\Delta_{\mathbf{D'}}}{1-u}$.
7: $\quad\quad\max_{j=0}^{n-1}|(\mathbf{L}_{1,0})_j| \leftarrow \frac{m_1-m_2}{m_1+m_2}$.
8: $\quad\quad C_1 = u\left[\frac{\sqrt{2}}{2}u^3 + \left(\frac{3\sqrt{2}}{2}+1\right)u^2 + \left(\frac{3\sqrt{2}}{2}+3\right)u + \frac{\sqrt{2}}{2} + 4\right]$.
9: $\quad\quad C_2' = \sqrt{2}u^4 + (3\sqrt{2}+2)u^3 + (3\sqrt{2}+6)u^2 + (\sqrt{2}+8)u + 4$.
10: $\quad\quad\Delta_{\mathbf{L}_{1,0}} \leftarrow \frac{C_1(m_1-m_2)+C_2'\Delta_{\mathbf{D'}}}{2(1-u)m_2-2(1+u)\Delta_{\mathbf{D'}}}$.
11: $\quad\quad$ffLDLErr($m_1, m_2, \Delta_{\mathbf{D}_{0,0}}, n/2$).
12: $\quad\quad$ffLDLErr($m_1, m_2, \Delta_{\mathbf{D}_{1,1}}, n/2$).
13: $\quad$**end if**
14: **end function**

We have $(\mathbf{G}_{\text{orig}})_{i,j} = \sum_{k=0}^{d-1}(\mathbf{S}_\ell)_{i,k}(\mathbf{S}_\ell^*)_{k,j}$, $0 \le i,j \le d-1$, for $\mathbf{G}_{\text{orig}} = \mathbf{S}_\ell\mathbf{S}_\ell^*$ in the coefficient domain. Therefore, $\mathbf{G}_{i,j} = \sum_{k=0}^{d-1}\text{FFT}(\mathbf{S}_\ell)_{i,k} \odot \text{FFT}(\mathbf{S}_\ell^*)_{k,j}$ in the FFT domain. By Theorem 4, we have $|\text{FFT}((\mathbf{S}_\ell)_{i,j})_k| \le \sqrt{N}\|(\mathbf{S}_\ell)_{i,j}\|$ for $0 \le k \le N-1$. From Algorithm 6.2 and Algorithm 6.3, we have $\|(\mathbf{S}_\ell)_i\| \le \sigma_\ell\sqrt{dN}$ for $0 \le i \le d-2$ and thus $\|(\mathbf{S}_\ell)_{i,j}\| \le \sigma_\ell\sqrt{dN}$ for $0 \le i \le d-2$, $0 \le j \le d-1$. Therefore, we have $|\text{FFT}((\mathbf{S}_\ell)_{i,j})_k| \le \sigma_\ell N\sqrt{d}$ for $0 \le i \le d-2$, $0 \le j \le d-1$, $0 \le k \le N-1$.

To estimate $\max_{k=0}^{N-1}|\text{FFT}((\mathbf{S}_\ell)_{d-1,j})_k|$, $0 \le j \le d-1$, we adapt the following heuristic estimation of $\|(\mathbf{S}_\ell)_{d-1}\|$ from [Pre15]. For fully length-reduced $(\mathbf{S}_\ell)_{d-1}$, we have:

$$\|(\mathbf{S}_\ell)_{d-1}\|^2 = \|(\tilde{\mathbf{S}}_\ell)_{d-1}\|^2 + \sum_{i=0}^{d-2}\|\mathbf{r}(\mathbf{S}_\ell)_i\|^2,$$

where $\mathbf{r}$ has coefficients in $[-1/2, 1/2]$ and $(\tilde{\mathbf{S}}_\ell)_{d-1}$ is orthogonal to $\text{Span}((\mathbf{S}_\ell)_0, \ldots, (\mathbf{S}_\ell)_{d-2})$. We have the heuristic estimation $\|\mathbf{r}(\mathbf{S}_\ell)_i\| \approx \sqrt{N/12}\|(\mathbf{S}_\ell)_i\|$ [HHP+03, Pre15]. Since $\|(\tilde{\mathbf{S}}_\ell)_{d-1}\| \le \sigma_\ell\sqrt{dN}$ for a (Mod)NTRU basis generated by Algorithm 6.2 and Algorithm 6.3 in Latte, we have $\|(\mathbf{S}_\ell)_{d-1}\| \le \sigma_\ell\sqrt{dN(1+(d-1)N/12)}$ and thus $\|(\mathbf{S}_\ell)_{d-1,j}\| \le \sigma_\ell\sqrt{dN(1+(d-1)N/12)}$, $|\text{FFT}((\mathbf{S}_\ell)_{d-1,j})_k| \le \sigma_\ell N\sqrt{d(1+(d-1)N/12)}$ for $0 \le j \le d-1$, $0 \le k \le N-1$.

Therefore, we have the following upper bounds of $|(\mathbf{G}_{i,j})_k|$ for $\mathbf{G}_{i,j} = \sum_{k=0}^{d-1}\text{FFT}(\mathbf{S}_\ell)_{i,k} \odot \text{FFT}(\mathbf{S}_\ell^*)_{k,j}$ in the FFT domain:

When $0 \le i, j \le d-2$**:**

$$|(\mathbf{G}_{i,j})_k| \le \sigma_\ell^2 N^2 d^2.$$

When $i = d-1, 0 \le j \le d-2$ **or** $0 \le i \le d-2, j = d-1$**:**

$$|(\mathbf{G}_{i,j})_k| \le \sigma_\ell^2 N^2 d^2 \sqrt{1 + (d-1)N/12}.$$

When $i = j = d-1$**:**

$$|(\mathbf{G}_{i,j})_k| \le \sigma_\ell^2 N^2 d^2 (1 + (d-1)N/12).$$

When $d = 2$, we have the absolute error bounds $\Delta_{\mathbf{G}_{0,0}}$ and $\Delta_{\mathbf{G}_{1,0}}$:

$\Delta_{\mathbf{G}_{0,0}}$ **(**$d = 2$**):** We have $(\mathbf{G}_{0,0})_k = |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2 \in \mathbb{R}^+$. By using the error analysis for real number arithmetic and Theorem 3, we get:

$$\begin{aligned}
\Delta_{\mathbf{G}_{0,0}} &\le \max_{k=0}^{N-1}[(1+u)[(1+u)(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k| + \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})})^2 \\
&\quad + (1+u)(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k| + \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})})^2] \\
&\quad - (|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2)] \\
&\le [(1+u)^2(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2) \\
&\le [(1+u)^2(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot 4\sigma_\ell^2 N^2.
\end{aligned}$$

$\Delta_{\mathbf{G}_{1,0}}$ **(**$d = 2$**):** By Theorem 3, we have:

$$\begin{aligned}
\Delta_{\mathbf{G}_{1,0}} \le \Delta_\pm \Bigg[&\max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|), \\
&\max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|), \\
&\Delta_\times\left(\max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})_k|, \max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})}\right), \\
&\Delta_\times\left(\max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})_k|, \max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})}\right)\Bigg] \\
&\le \Delta_\pm(2\sigma_\ell^2 N^2\sqrt{1+N/12}, 2\sigma_\ell^2 N^2\sqrt{1+N/12}, \Delta, \Delta),
\end{aligned}$$

where

$$\Delta = \Delta_\times(\sigma_\ell N\sqrt{2+N/6}, \sigma_\ell N\sqrt{2}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{2+N/6}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{2}).$$

When $d = 3$, because the upper bound of $|\mathrm{FFT}((\mathbf{S}_\ell)_{1,j})_k|$ is equivalent to the upper bound of $|\mathrm{FFT}((\mathbf{S}_\ell)_{0,j})_k|$, $0 \le j \le d-1$, $0 \le k \le N-1$, we have the upper bound of $\Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,j})}$ equivalent to the upper bound of $\Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,j})}$ by Theorem 3. Thus, the upper bound of

$\Delta_{\mathbf{G}_{1,1}}$ is equivalent to the upper bound of $\Delta_{\mathbf{G}_{0,0}}$, and the upper bound of $\Delta_{\mathbf{G}_{2,1}}$ is equivalent to the upper bound of $\Delta_{\mathbf{G}_{2,0}}$. Similar to the case when $d = 2$, we have the absolute error bounds $\Delta_{\mathbf{G}_{0,0}}$, $\Delta_{\mathbf{G}_{1,0}}$, $\Delta_{\mathbf{G}_{2,0}}$:

$\Delta_{\mathbf{G}_{0,0}}$ ($d = 3$): We have $(\mathbf{G}_{0,0})_k = |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k|^2 \in \mathbb{R}^+$. Therefore, we get:

$$\begin{aligned}
\Delta_{\mathbf{G}_{0,0}} &\leq \max_{k=0}^{N-1}[(1+u)[(1+u)[(1+u)(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k| + \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})})^2 \\
&\quad + (1+u)(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k| + \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})})^2] \\
&\quad + (1+u)(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k| + \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})})^2] \\
&\quad - (|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k|^2)] \\
&\leq [(1+u)^3(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k|^2 + |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k|^2) \\
&\quad + [(1+u)^2(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot \max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k|^2 \\
&\leq [(1+u)^3(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot 6\sigma_\ell^2 N^2 + [(1+u)^2(1+\delta_{\mathrm{FFT}})^2 - 1] \cdot 3\sigma_\ell^2 N^2.
\end{aligned}$$

$\Delta_{\mathbf{G}_{1,0}}$ ($d = 3$): We have:

$$\begin{aligned}
\Delta_{\mathbf{G}_{1,0}} \leq \Delta_\pm \Bigg[& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*| + |\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|), \\
& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,2})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k^*|), \\
& \Delta_\pm \Bigg[\max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|), \\
& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|), \\
& \Delta_\times \left(\max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})_k|, \max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,0})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})} \right), \\
& \Delta_\times \left(\max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})_k|, \max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,1})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})} \right) \Bigg], \\
& \Delta_\times \left(\max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{1,2})_k|, \max_{k=0}^{N-1} |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{1,2})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})} \right) \Bigg] \\
\leq \Delta_\pm [& 6\sigma_\ell^2 N^2, 3\sigma_\ell^2 N^2, \Delta_\pm(3\sigma_\ell^2 N^2, 3\sigma_\ell^2 N^2, \Delta, \Delta), \Delta],
\end{aligned}$$

where

$$\Delta = \Delta_\times(\sigma_\ell N\sqrt{3}, \sigma_\ell N\sqrt{3}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{3}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{3}).$$

$\Delta_{\mathbf{G}_{2,0}}$ **($d = 3$):** We have:

$$\begin{aligned}
\Delta_{\mathbf{G}_{2,0}} \leq \Delta_{\pm}\Bigg[& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{2,0})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*| + |\mathrm{FFT}((\mathbf{S}_\ell)_{2,1})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|), \\
& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{2,2})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k^*|), \\
& \Delta_{\pm}\Bigg[\max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{2,0})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|), \\
& \max_{k=0}^{N-1}(|\mathrm{FFT}((\mathbf{S}_\ell)_{2,1})_k| \cdot |\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|), \\
& \Delta_{\times}\left(\max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{2,0})_k|, \max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{2,0})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,0})}\right), \\
& \Delta_{\times}\left(\max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{2,1})_k|, \max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{2,1})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,1})}\right)\Bigg], \\
& \Delta_{\times}\left(\max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{2,2})_k|, \max_{k=0}^{N-1}|\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})_k^*|, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{2,2})}, \Delta_{\mathrm{FFT}((\mathbf{S}_\ell)_{0,2})}\right)\Bigg] \\
& \leq \Delta_{\pm}[6\sigma_\ell^2 N^2\sqrt{1+N/6}, 3\sigma_\ell^2 N^2\sqrt{1+N/6}, \\
& \Delta_{\pm}(3\sigma_\ell^2 N^2\sqrt{1+N/6}, 3\sigma_\ell^2 N^2\sqrt{1+N/6}, \Delta, \Delta), \Delta],
\end{aligned}$$

where

$$\Delta = \Delta_{\times}(\sigma_\ell N\sqrt{3+N/2}, \sigma_\ell N\sqrt{3}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{3+N/2}, \delta_{\mathrm{FFT}} \cdot \sigma_\ell N\sqrt{3}).$$

In addition, to use Algorithm 6.8 to compute $\Delta_{\mathbf{D}}$ and $\Delta_{\mathbf{L}}$ for non-root nodes, we need $\max_{j=0}^{N-1}(\mathbf{D}_{i,i})_j$ and $\min_{j=0}^{N-1}(\mathbf{D}_{i,i})_j$ for each $\mathbf{D}_{i,i} \in (\mathbb{R}^+)^N$, $0 \leq i \leq d-1$, at the root. Here, we analyse these bounds of $(\mathbf{D}_{i,i})_j$ for both $d = 2$ and $d = 3$.

When $d = 2$: Since $(\mathbf{D}_{0,0})_j = (\mathbf{G}_{0,0})_j$, we have $(\mathbf{D}_{0,0})_j \leq \max_{k=0}^{N-1}|(\mathbf{G}_{0,0})_k| \leq 4\sigma_\ell^2 N^2$. By Theorem 16, $(\mathbf{D}_{1,1})_j = q^2/(\mathbf{D}_{0,0})_j$ and thus $(\mathbf{D}_{1,1})_j \geq \frac{q^2}{4\sigma_\ell^2 N^2}$. Similarly, since $(\mathbf{D}_{1,1})_j = (\mathbf{G}_{1,1})_j - |(\mathbf{G}_{1,0})_j|^2/(\mathbf{D}_{0,0})_j$ where $(\mathbf{G}_{1,1})_j, (\mathbf{D}_{0,0})_j, (\mathbf{D}_{1,1})_j \in \mathbb{R}^+$, we have $(\mathbf{D}_{1,1})_j \leq (\mathbf{G}_{1,1})_j \leq \max_{k=0}^{N-1}|(\mathbf{G}_{1,1})_k| \leq 4\sigma_\ell^2 N^2(1+N/12)$. Then, we have $(\mathbf{D}_{0,0})_j = q^2/(\mathbf{D}_{1,1})_j \geq \frac{q^2}{4\sigma_\ell^2 N^2(1+N/12)}$ by Theorem 16.

When $d = 3$: Similar to the case when $d = 2$, we have $(\mathbf{D}_{0,0})_j = (\mathbf{G}_{0,0})_j \leq 9\sigma_\ell^2 N^2$ and $(\mathbf{D}_{0,0})_j \leq (\mathbf{G}_{1,1})_j \leq 9\sigma_\ell^2 N^2$. By using the empirical observation that $(\mathbf{D}_{2,2})_j \in [0,1]$ in our experiment for Latte parameter sets in Table 6.1, we get $(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j \geq q^2$ by using $(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j(\mathbf{D}_{2,2})_j = q^2$ from Theorem 16. Thus, we get $(\mathbf{D}_{0,0})_j, (\mathbf{D}_{1,1})_j \geq \frac{q^2}{9\sigma_\ell^2 N^2}$. For $(\mathbf{D}_{2,2})_j$, we have $(\mathbf{D}_{2,2})_j \leq 1$ from our observation. By Theorem 16, we have $(\mathbf{D}_{2,2})_j = \frac{q^2}{(\mathbf{D}_{0,0})_j(\mathbf{D}_{1,1})_j} \geq \frac{q^2}{81\sigma_\ell^4 N^4}$.

By using the Latte parameter sets in Table 6.1, we can compute the numerical values of the *relative* error δ_σ and the *absolute* error $\Delta_{\mathbf{L}}$ in our optimised Latte scheme by applying the error analysis above. We use the parameters of basis $\mathbf{S}_0$ when computing δ_σ, $\Delta_{\mathbf{L}}$ for $d = 2$, and we use the parameters of the delegated basis $\mathbf{S}_1$ when computing δ_σ, $\Delta_{\mathbf{L}}$ for $d = 3$, respectively. Note that only Latte-3 and 4 need the ffLDL algorithm of $d = 3$ for the delegated basis $\mathbf{S}_1$, since Latte-1 and 2 are essentially single-level IBE schemes without delegation. Results of δ_σ and $\Delta_{\mathbf{L}}$ for Latte parameter sets with different floating-point arithmetic errors u are shown in Table 6.3 and Table 6.4 for $d = 2$ and $d = 3$, respectively. Columns in Table 6.3 and Table 6.4 with parameter m are the errors of non-root nodes whose ancestor is the m-th child of the root.

Table 6.3: Numerical Values of δ_σ and $\Delta_{\mathbf{L}}$ for Latte Parameter Sets ($d = 2$).

	$u = 2^{-96}$				
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}}$ ($m = 0$)	δ_σ ($m = 0$)	$\Delta_{\mathbf{L}}$ ($m = 1$)	δ_σ ($m = 1$)
Latte-1	2^{-22}	2^{-46}	2^{-47}	2^{-17}	2^{-18}
Latte-2	2^{-15}	2^{-41}	2^{-42}	2^{-9}	2^{-10}
Latte-3	2^{-22}	2^{-46}	2^{-47}	2^{-17}	2^{-18}
Latte-4	2^{-15}	2^{-41}	2^{-42}	2^{-9}	2^{-10}
	$u = 2^{-112}$				
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}}$ ($m = 0$)	δ_σ ($m = 0$)	$\Delta_{\mathbf{L}}$ ($m = 1$)	δ_σ ($m = 1$)
Latte-1	2^{-38}	2^{-62}	2^{-63}	2^{-33}	2^{-34}
Latte-2	2^{-31}	2^{-57}	2^{-58}	2^{-25}	2^{-26}
Latte-3	2^{-38}	2^{-62}	2^{-63}	2^{-33}	2^{-34}
Latte-4	2^{-31}	2^{-57}	2^{-58}	2^{-25}	2^{-26}
	$u = 2^{-128}$				
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}}$ ($m = 0$)	δ_σ ($m = 0$)	$\Delta_{\mathbf{L}}$ ($m = 1$)	δ_σ ($m = 1$)
Latte-1	2^{-55}	2^{-78}	2^{-79}	2^{-49}	2^{-50}
Latte-2	2^{-48}	2^{-74}	2^{-75}	2^{-41}	2^{-42}
Latte-3	2^{-55}	2^{-78}	2^{-79}	2^{-49}	2^{-50}
Latte-4	2^{-48}	2^{-74}	2^{-75}	2^{-41}	2^{-42}

From Table 6.3 and Table 6.4, for the same Latte parameter set, $\Delta_{\mathbf{L}}$ and δ_σ are linearly proportional to the relative error u of floating-point arithmetic at every node (including both the root and non-root nodes) of the ffLDL tree in both cases when $d = 2$ and $d = 3$.

However, both $\Delta_{\mathbf{L}}$ and δ_σ have big gaps between the cases when $m < d-1$ and $m = d-1$ for all Latte parameter sets. For example, both $\Delta_{\mathbf{L}}$ and δ_σ have about 30 bits difference between $m = 0$ and $m = 1$ in Table 6.3 when $d = 2$, and the gap is more than 45 bits between $m \in \{0, 1\}$ and $m = 2$ in Table 6.4 when $d = 3$. The reason for these gaps is that when computing $\Delta_{\mathbf{D}_{d-1,d-1}}$ at the root in Section 6.3.1, the minimal value of the denominator is estimated by using the smallest value among coordinates of some vector minus the maximal absolute error among coordinates of the same vector. The result is very likely to be much smaller than one could get in practice and it may significantly

Table 6.4: Numerical Values of δ_σ and $\Delta_{\mathbf{L}}$ for Latte Parameter Sets ($d = 3$).

	$u = 2^{-240}$								
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}_{2,0}}$ (root)	$\Delta_{\mathbf{L}_{2,1}}$ (root)	$\Delta_{\mathbf{L}}$ $(m = 0)$	δ_σ $(m = 0)$	$\Delta_{\mathbf{L}}$ $(m = 1)$	δ_σ $(m = 1)$	$\Delta_{\mathbf{L}}$ $(m = 2)$	δ_σ $(m = 2)$
Latte-3	2^{-132}	2^{-129}	2^{-32}	2^{-171}	2^{-172}	2^{-123}	2^{-124}	2^{-75}	2^{-76}
Latte-4	2^{-123}	2^{-119}	2^{-15}	2^{-166}	2^{-167}	2^{-113}	2^{-114}	2^{-61}	2^{-62}
	$u = 2^{-256}$								
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}_{2,0}}$ (root)	$\Delta_{\mathbf{L}_{2,1}}$ (root)	$\Delta_{\mathbf{L}}$ $(m = 0)$	δ_σ $(m = 0)$	$\Delta_{\mathbf{L}}$ $(m = 1)$	δ_σ $(m = 1)$	$\Delta_{\mathbf{L}}$ $(m = 2)$	δ_σ $(m = 2)$
Latte-3	2^{-149}	2^{-145}	2^{-49}	2^{-188}	2^{-189}	2^{-140}	2^{-141}	2^{-92}	2^{-93}
Latte-4	2^{-140}	2^{-136}	2^{-31}	2^{-182}	2^{-183}	2^{-130}	2^{-131}	2^{-78}	2^{-79}
	$u = 2^{-272}$								
	$\Delta_{\mathbf{L}_{1,0}}$ (root)	$\Delta_{\mathbf{L}_{2,0}}$ (root)	$\Delta_{\mathbf{L}_{2,1}}$ (root)	$\Delta_{\mathbf{L}}$ $(m = 0)$	δ_σ $(m = 0)$	$\Delta_{\mathbf{L}}$ $(m = 1)$	δ_σ $(m = 1)$	$\Delta_{\mathbf{L}}$ $(m = 2)$	δ_σ $(m = 2)$
Latte-3	2^{-164}	2^{-161}	2^{-64}	2^{-203}	2^{-204}	2^{-155}	2^{-156}	2^{-107}	2^{-108}
Latte-4	2^{-155}	2^{-151}	2^{-47}	2^{-198}	2^{-199}	2^{-145}	2^{-146}	2^{-93}	2^{-94}

increase $\Delta_{\mathbf{D}_{d-1,d-1}}$. This larger bound $\Delta_{\mathbf{D}_{d-1,d-1}}$ will then be propagated to the computation of $\Delta_{\mathbf{L}}$ and δ_σ in non-root nodes whose ancestor is the $(d-1)$-th child of the root when using Algorithm 6.8.

In addition, there is near 100 bits difference between $\Delta_{\mathbf{L}_{1,0}}$ (or $\Delta_{\mathbf{L}_{2,0}}$) and $\Delta_{\mathbf{L}_{2,1}}$ in Table 6.4 when $d = 3$. This is because the maximal value of the numerator $(\mathbf{G}_{2,1})_j - (\mathbf{G}_{2,0})_j(\mathbf{L}_{1,0})_j^*$ in $(\mathbf{L}_{2,1})_j$ may be significantly overestimated by using $\max_{k=0}^{N-1} |(\mathbf{G}_{2,1})_k| + \left(\max_{k=0}^{N-1} |(\mathbf{G}_{2,0})_k|\right)\left(\max_{k=0}^{N-1} |(\mathbf{L}_{1,0})_k|\right)$ in Section 6.3.2.

Furthermore, there is a big gap between our provable error analysis results and the errors one can get in practice. For both $d = 2$ and $d = 3$, we measure the δ_σ in Latte-3 by comparing the σ computed with $p = \{72, 80, 96, 128, 256\}$ bit floating-point precision against the "accurate" σ computed with 1024 bit high precision when using the same random seed in our experiment. Results of the maximal δ_σ among 1000 different random seeds are shown in Table 6.5. On the one hand, the results in Table 6.5 verify that δ_σ is linearly proportional to the relative error u of floating-point arithmetic for Latte-3 in both cases when $d = 2$ and $d = 3$ in practice. On the other hand, Table 6.5 shows the difference between δ_σ and u in practice is 2 bits when $d = 2$ and 7 bits when $d = 3$ *at most* for Latte-3, which is much smaller than 49–78 bits ($d = 2$) and 67–164 bits ($d = 3$) from the numerical results of our provable error analysis in Table 6.3 and Table 6.4. We will leave improving our theoretical error bounds in order to reduce the huge gap from practice as future works.

Table 6.5: δ_σ in Practice for Latte-3.

Precision	δ_σ $(d = 2)$	δ_σ $(d = 3)$
72	2^{-69}	2^{-65}
80	2^{-78}	2^{-75}
96	2^{-94}	2^{-92}
128	2^{-126}	2^{-123}
256	2^{-254}	2^{-251}

Our provable error analysis has another issue that it still contains heuristic components, namely:

1. The heuristic estimation of $\|(\mathbf{S}_\ell)_{d-1}\|$ from [HHP+03, Pre15].
2. The empirical observation $(\mathbf{D}_{2,2})_j \in [0, 1]$, $0 \le j \le N-1$, at the root when $d = 3$ in our experiment for Latte parameter sets in Table 6.1.

Note that the second heuristic above is likely to be true only for a delegated ModNTRU basis in Latte HIBE, and should not be applied to other ModNTRU based schemes such as [CKKS19, CPS+20], where a fresh ModNTRU basis is directly generated as the master private key during the Master KeyGen. This is because in schemes such as [CKKS19,

CPS+20], the σ_0 used for the Master KeyGen is chosen to keep the Gram-Schmidt norm $\|\tilde{\mathbf{S}}_i\|$ close to each other between each Gram-Schmidt vector $\tilde{\mathbf{S}}_i$ of the generated secret basis $\mathbf{S}$, similar to how σ_0 is chosen in the DLP IBE [DLP14]. However, in the Latte HIBE, $\sigma_\ell = (\eta_\epsilon(\mathbb{Z})/\sqrt{2\pi})\sqrt{(\ell+1)N}\sigma_{\ell-1}$ due to the GPV sampling condition [GPV08, ETS19] (see Section 3.2.1), and the Euclidean norms of the last N Gram-Schmidt vectors are likely to be much smaller compared to the first $(d-1)N$ Gram-Schmidt vectors in a delegated basis, as discussed in Section 6.1.2. Since the bounds of $(\mathbf{D}_{2,2})_j$ are related to the Euclidean norms of the last N Gram-Schmidt vectors by Lemma 4, $(\mathbf{D}_{2,2})_j$ is likely to be small only for the delegated basis in Latte HIBE but not for the master private keys in other ModNTRU based schemes. We will leave the proof of these heuristic assumptions to future works.

6.4 Evaluation

The first published specification of Latte is [ETS19]. However, it only provided the Encrypt and Decrypt performance results. The performance results in [ETS19] were obtained from an AMD A10-6700 CPU at 3.7GHz, which has a different CPU frequency compared to our benchmark platform with an Intel i7-7700K CPU at 4.2GHz. Therefore, we scale all the performance results from [ETS19] to 4.2GHz in this section. The Encrypt/Decrypt performance results of the original Latte implementation [ETS19] are displayed in Table 6.6, scaled and converted into op/s at 4.2GHz.

Table 6.6: Proof of Concept Latte Performance Results (op/s) from [ETS19] (Scaled to 4.2GHz).

	$\ell = 1$		$\ell = 2$	
Set	**Enc**	**Dec**	**Enc**	**Dec**
Latte-1	2911	2987	-	-
Latte-2	1335	1351	-	-
Latte-3	1892	1774	1455	1474
Latte-4	709	668	568	541

Here, we give the first full performance results for our optimised variant of Latte, including KeyGen, Extract, and Delegate. We employ the `gmp` [GT15], `mpfr` [FHL+07], and `mpc` [EGTZ18] libraries for multiprecision integer, floating-point, and complex number arithmetic, respectively. The precision of floating-point and complex numbers in our implementation is $\lambda = 256$ bits. In addition, we use the AES-256 CTR mode with hardware AES-NI instructions [Gue09] as the pseudorandom generator, and use SHAKE-256 [NIS15] as the KDF in Latte Encrypt and Decrypt. The performance results have been

obtained from a desktop machine with an Intel i7-7700K CPU at 4.2GHz, with both hyper-threading and TurboBoost disabled. We use the gcc 11.2.0 compiler with compiling options `-O3 -march=native` enabled. Results are given in Table 6.7.

Table 6.7: Our Optimised Latte Performance Results (op/s) at 4.2GHz.

		$\ell = 1$				$\ell = 2$		
Set	**KeyGen**	**Ext**	**Enc**	**Dec**	**Del**	**Ext**	**Enc**	**Dec**
Latte-1	4.7	5.8	12187.7	10424.9	-	-	-	-
Latte-2	1.4	2.0	5776.4	4935.1	-	-	-	-
Latte-3	3.6	5.8	6334.7	5240.3	1.0	3.7	5234.7	4396.8
Latte-4	1.0	2.0	3094.3	2580.9	0.4	1.3	2569.2	2166.5

As expected, the KeyGen, Extract, and Delegate processes are the most time consuming components of the scheme, and this increases as security and therefore lattice dimension increase. The trend down the hierarchical levels is that the Extract, Encrypt, and Decrypt all become more time consuming as hierarchical level increases. For Extract in Latte-3 and 4, this corresponds to about 35% decrease in op/s from level 1 to level 2. On the other hand, for the Encrypt and Decrypt, our implementation is 3.0x–4.5x faster compared to the previous performance results from [ETS19]. The speedup might be due to: (1) We change the distribution of the ephemeral private keys from discrete Gaussian distribution to binomial distribution. (2) We only perform NTT for the ephemeral private keys and $\mathbf{m}$ during the Encrypt and Decrypt, since other inputs are already in the NTT domain. In addition, our optimised Latte Delegate only takes about 1–2.5 seconds on a desktop machine at 4.2GHz, which is practical and much faster than the estimated run-time (in the order of minutes) for the Delegate in the un-optimised variants of Latte [ETS19].

Comparison to DLP IBE Performance results of the single-level DLP IBE scheme from [ETS19] (converted to op/s at 4.2GHz) are given in Table 6.8. Since the decryption in the DLP IBE did not include ciphertext validation, for a fair comparison with Latte, we use the sum of DLP encryption and decryption run-time to compute the op/s of decryption in Table 6.8. We focus on the comparison between Latte-1 and DLP-3, since the sizes of parameters N and q are similar. The KeyGen speed of our Latte-1 implementation is similar to DLP-3, and the Encrypt/Decrypt speed is 4.6x–6.4x faster in our implementation. However, the speed of Extract in our implementation is very slow. For example, our Latte-1 Extract implementation is about 78x slower than DLP-3 extraction. This is mainly because we use 256-bit multiprecision in the key sub-algorithms (Tree and ffSampling) and in the variant of COSAC sampler used by our implementation, while it is sufficient to use less than 64-bit precision in DLP extraction [MSO17].

Table 6.8: Performance Results (op/s) for the DLP IBE Scheme from [ETS19] (Scaled to 4.2GHz).

Set	**Security**	n	$\log_2 q$	**KeyGen**	**Ext**	**Enc**	**Dec**
DLP-0	80	512	22	14.7	873.2	8731.8	6202.9
DLP-3	192	1024	22	4.9	454.1	2639.8	1621.6

Chapter 7

Conclusion and Discussion

In this thesis, we present fast, compact, and constant-time (FACCT) zero-centered discrete Gaussian sampler over integers, by implementing the Bernoulli sampler in the binary sampling scheme with a constant-time $\exp(x)$ polynomial approximation. Our implementation is faster than previous countermeasures [PBY17, EFGT17], more compact than both the qTesla-R1 [BAA+17] and R2 [ABB+19] samplers, and outperforms the bitslicing convolution scheme [KRR+18] in both timing and memory consumption. Our implementation techniques are also independent of the standard deviation, and have good flexibility and performance in various applications. In addition, we show the smaller base sampler deviations for the convolution schemes by adapting the Rényi divergence.

We generalise the idea from [Dev86] and present a compact and scalable arbitrary-centered discrete Gaussian sampling scheme over integers. Our scheme performs rejection sampling on rounded samples from a continuous normal distribution, which does not rely on any discrete Gaussian sampling implementations. We show that our scheme maintains good performance for $\sigma \in [2, 2^{20}]$ and needs no pre-computations related to any specific σ, which is suitable to implement applications that requires sampling from multiple different σ. In addition, we provide concrete rejection rate and error analysis of our scheme. The performance of our scheme heavily relies on the underlying continuous Gaussian sampling algorithm. However, the Box-Muller sampler [HLS18, ZCHW17] employed in our implementation does not have the fastest sampling speed compared to other algorithms according to a survey [TLLV07]. The main reason behind the choice of the continuous Gaussian sampler in our implementation is because the Box-Muller sampler is very simple to implement in constant-time [HLS18]. If the side-channel perspective is not a concern, one may employ other more efficient non-constant time algorithms from the survey [TLLV07] to achieve a faster implementation of our scheme.

We adapt the NTRUSolve function and the FFT sampling procedure from (Mod)Falcon [PFH+17, CPS+20] in order to optimise the implementation of the Latte HIBE scheme [ETS19]. For the lattice basis in Latte, we provide an optimised Fast Fourier $\mathbf{LDL}^*$ decomposition (ffLDL) algorithm, which is one of the key subroutines used by the FFT sampling procedure [PFH+17]. Our optimised ffLDL algorithm is more than 70% faster than a generic naive ffLDL implementation under 256-bit floating-point arithmetic precision for Latte parameter sets. In addition, we provide the first provable theoretical error analysis of the ffLDL algorithm and compute the numerical values of the precision bounds based on the Latte parameter sets. We also adapt both the FACCT and the COSAC integer discrete Gaussian samplers in our optimised Latte implementation. Combining all techniques above, we provide the first complete implementation result of the Latte HIBE. We show that the Delegate function in our optimised Latte implementation will only take a few seconds on a desktop machine, which is significantly faster than the estimated run-time (in the order of minutes) in [ETS19]. However, our current implementation is not constant-time, since the arithmetic in KeyGen, Delegate, and Extract relies on multiprecision libraries [EGTZ18, FHL+07, GT15]. In addition, the multiprecision arithmetic is also the bottleneck in the Latte Extract, and we find that the speeds of Latte KeyGen, Delegate, and Extract are linearly proportional to the precision in our experiment. Furthermore, our provable theoretical precision upper bounds of the ffLDL algorithm might be significantly overestimated, as discussed in Section 6.3.4.

7.1 Future Works

ARM Cortex-M4 Implementations The main target of our techniques in this thesis is the powerful Intel platform. However, it remains an interesting question whether our techniques can also be adapted to more resource-constrained platforms, such as the ARM Cortex-M4, which is currently used in many embedded devices. There are two main challenges of adapting our techniques on Cortex-M4 platforms:

- The implementations of our FACCT and COSAC samplers in Chapter 4 and 5 heavily rely on (constant-time) hardware double precision floating-point instructions. However, Cortex-M4 platforms may *optionally*[1] provide the floating-point unit i.e. hardware floating-point instructions, supporting only the single precision[2] floating-point arithmetic. Therefore, on Cortex-M4 platforms, our current FACCT and COSAC implementations need to be significantly modified in order to avoid

[1] https://developer.arm.com/documentation/ddi0439/b/Floating-Point-Unit.
[2] https://developer.arm.com/documentation/ddi0439/b/Floating-Point-Unit/About-the-FPU.

the potentially non-constant time software routines [OSHG19] from the C run-time library computing the double precision floating-point arithmetic. One approach to realise this is adapting the fixed-point arithmetic from the GALACTICS [BBE+19], which only involves integers.

- Although the technical report of the original Latte scheme [ETS19] demonstrated that the Encrypt and Decrypt of the original Latte implementation have run-time in the order of milliseconds on an ARM1176 embedded processor at 700MHz, however, our benchmark results on the Intel platform in Section 6.4 indicate that the KeyGen, Delegate, and Extract of our optimised Latte implementation are computation intensive. Since our optimised Latte implementation shares several key subroutines (notably NTRUSolve, ffLDL, and ffSampling) with the Falcon digital signature scheme [PFH+17], we may adapt optimisation techniques from the Falcon Cortex-M4 implementation [Por19]. However, the challenge is how to adapt the optimisation techniques from [Por19] for the much larger parameters and potentially higher precision requirements of Latte compared to Falcon [PFH+17].

Power Analysis Resistant FACCT Sampler Our FACCT sampler is designed to resist timing/cache side-channel attacks but not power analysis attacks. Very recently, Marzougui et al. reported that our FACCT sampler and its variant used by the unmasked GALACTICS digital signature implementation [BBE+19] are insecure against machine learning based power analysis side-channel attacks [MWG+21]. For applications that require power analysis side-channel resistance, it is an interesting open problem to design a practical power analysis resistant variant of the FACCT sampler with a small efficiency overhead over the original FACCT algorithm.

Complete Provable Precision Analysis of Latte HIBE As mentioned in Section 6.3, in order to adapt the Rényi divergence argument from Falcon [PFH+17, HPRR20] to provide the provable arithmetic precision bounds of the FFT sampling procedure under the Latte HIBE settings, in addition to the analysis of δ_σ in Section 6.3, we also need the upper bound of the absolute error Δ_c among the center c for the integer discrete Gaussian sampling subroutine in ffSampling. We believe this can be done using a similar approach as our ffLDL error analysis in Section 6.3 to analyse the absolute errors of the ffSampling algorithm and leave it to future work.

Improved Precision Analysis for Latte HIBE in Practice Although we may adapt our provable precision analysis results in the Latte implementation, however, as discussed in Section 6.3.4, our provable precision upper bounds might be significantly overestimated at the root of the ffLDL tree. These overestimated errors from the root will

then be propagated to the error analysis of other non-root nodes in the tree. Therefore, one open problem is to sharpen our theoretical error bounds for ffLDL to reduce the large gap in the error bounds we obtained among the tree values. Another approach could be to use the empirical experimental results instead for the error bounds of the root in the ffLDL tree, and then use Algorithm 6.8 to work out the errors of non-root nodes. Since the run-time speed of the Latte algorithms are linearly proportional to the floating-point arithmetic precision, we can also accelerate the run-time speed of our Latte implementation by improving the precision analysis.

Fully Constant-time Latte HIBE Implementation As mentioned above, our current Latte HIBE implementation is not constant-time. To make the Latte HIBE implementation constant-time, we need to fix the floating-point arithmetic precision in our constant-time implementation and employ suitable constant-time floating-point arithmetic operations i.e. replacing those non-constant time arithmetic operations from multiprecision libraries [EGTZ18, FHL+07, GT15]. For the constant-time floating-point arithmetic operations, we can adapt techniques from the constant-time Falcon implementation [PRR19].

References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572. Springer, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2010.

[ABB+19] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qTESLA. *IACR Cryptology ePrint Archive*, 2019:85, 2019.

[ABG+21] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In *CT-RSA*, volume 12704 of *Lecture Notes in Computer Science*, pages 227–251. Springer, 2021.

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. *IACR Cryptology ePrint Archive*, 2015:1092, 2015.

[Aum19] Jean-Philippe Aumasson. Guidelines for low-level cryptography software. https://github.com/veorq/cryptocoding, 2019. Accessed: 2020-01-28.

[BAA+17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. Submission to NIST's post-quantum project: lattice-based digital signature scheme qTESLA. https://qtesla.org/, 2017. Accessed: 2018-11-03.

[BBE+19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In *CCS*, pages 2147–2164. ACM, 2019.

[BC07] Nicolas Brisebarre and Sylvain Chevillard. Efficient polynomial l-approximations. In *IEEE Symposium on Computer Arithmetic*, pages 169–176. IEEE Computer Society, 2007.

[BCNS15] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society, 2015.

[BEP+21] Pauline Bert, Gautier Eberhart, Lucas Prabel, Adeline Roux-Langlois, and Mohamed Sabt. Implementation of lattice trapdoors on modules and applications. In *PQCrypto*, volume 12841 of *Lecture Notes in Computer Science*, pages 195–214. Springer, 2021.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BFRS18] Pauline Bert, Pierre-Alain Fouque, Adeline Roux-Langlois, and Mohamed Sabt. Practical implementation of ring-SIS/LWE based signature and IBE. In *PQCrypto*, volume 10786 of *Lecture Notes in Computer Science*, pages 271–291. Springer, 2018.

[BHK+22] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. Neon NTT: faster dilithium, kyber, and saber on cortex-a72 and apple M1. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):221–244, 2022.

[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 *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345. Springer, 2016.

[BJM+20] Nicolas Brisebarre, Mioara Joldes, Jean-Michel Muller, Ana-Maria Nanes, and Joris Picot. Error analysis of some operations involved in the cooley-tukey fast fourier transform. *ACM Trans. Math. Softw.*, 46(2):11:1–11:27, 2020.

[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 (1)*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015.

[CG50] Cramer and Gabriel. *Introduction a l'analyse des lignes courbes algebriques par Gabriel Cramer.* chez les freres Cramer & Cl. Philibert, 1750.

[CHK+21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for ntt-unfriendly rings new speed records for saber and NTRU on cortex-m4 and AVX2. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):159–188, 2021.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552. Springer, 2010.

[CJL10] Sylvain Chevillard, Mioara Joldes, and Christoph Quirin Lauter. Sollya: An environment for the development of numerical codes. In *ICMS*, volume 6327 of *Lecture Notes in Computer Science*, pages 28–31. Springer, 2010.

[CKKS19] Jung Hee Cheon, Duhyeong Kim, Taechan Kim, and Yongha Son. A new trapdoor over Module-NTRU lattice and its application to ID-based encryption. *IACR Cryptol. ePrint Arch.*, 2019:1468, 2019.

[CLJ18] Sylvain Chevillard, Christoph Lauter, and Mioara Joldes. Users‘ manual for the sollya tool. https://www.sollya.org/releases/sollya-7.0/sollya-7.0.pdf, 2018. Accessed: 2021-10-11.

[CPS+20] Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. Modfalcon: Compact signatures based on module-ntru lattices. In *AsiaCCS*, pages 853–866. ACM, 2020.

[CT65] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[DB15] Chaohui Du and Guoqiang Bai. Towards efficient discrete Gaussian sampling for lattice-based cryptography. In *FPL*, pages 1–6. IEEE, 2015.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.

[Dev86] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, USA, 1986.

[DFW20] Yusong Du, Baoying Fan, and Baodian Wei. Arbitrary-centered discrete gaussian sampling over the integers. In *ACISP*, volume 12248 of *Lecture Notes in Computer Science*, pages 391–407. Springer, 2020.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng. Commun. Comput.*, 25(3):159–180, 2014.

[DLL+17] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - dilithium: Digital signatures from module lattices. *IACR Cryptology ePrint Archive*, 2017:633, 2017.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2014.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2012.

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *ISSAC*, pages 191–198. ACM, 2016.

[DWZ19] Yusong Du, Baodian Wei, and Huang Zhang. A rejection sampling algorithm for off-centered discrete Gaussian distributions over the integers. *SCIENCE CHINA Information Sciences*, 62(3):39103:1–39103:3, 2019.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *CCS*, pages 1857–1874. ACM, 2017.

[EGTZ18] Andreas Enge, Mickaël Gastineau, Philippe Théveny, and Paul Zimmermann. *mpc — A library for multiprecision complex arithmetic with exact rounding*. INRIA, 1.1.0 edition, January 2018. http://mpc.multiprecision.org/.

[ETS19] ETSI. Quantum-Safe Identity-based Encryption. Technical report, The European Telecommunications Standards Institute, Sophia-Antipolis, France, 2019.

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2):13, 2007.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[Fog17] Agner Fog. VCL C++ vector class library. www.agner.org/optimize/vectorclass.pdf, 2017. Accessed: 2019-08-01.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[GM18] Nicholas Genise and Daniele Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In *EUROCRYPT (1)*, volume 10820 of *Lecture Notes in Computer Science*, pages 174–203. Springer, 2018.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. ACM, 2008.

[GS66] W. Morven Gentleman and G. Sande. Fast fourier transforms: for fun and profit. In *AFIPS Fall Joint Computing Conference*, volume 29 of *AFIPS Conference Proceedings*, pages 563–578. AFIPS / ACM / Spartan Books, Washington D.C., 1966.

[GT15] Torbjrn Granlund and Gmp Development Team. *GNU MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited, London, GBR, 2015.

[Gue09] Shay Gueron. Intel's new AES instructions for enhanced performance and security. In *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2009.

[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*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

[HKR+18] James Howe, Ayesha Khalid, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. On practical discrete Gaussian samplers for lattice-based cryptography. *IEEE Trans. Computers*, 67(3):322–334, 2018.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2002.

[HLS18] Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded Gaussians - fast and secure constant-time sampling for lattice-based crypto. In *Public Key Cryptography (2)*, volume 10770 of *Lecture Notes in Computer Science*, pages 728–757. Springer, 2018.

[HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In *PQCrypto*, volume 12100 of *Lecture Notes in Computer Science*, pages 53–71. Springer, 2020.

[Int19] Intel. Intel intrinsics guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide/, 2019. Accessed: 2019-03-06.

[Kar16] Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1):3:1–3:14, 2016.

[KHR+18] Ayesha Khalid, James Howe, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. Compact, scalable, and efficient discrete gaussian samplers for lattice-based cryptography. In *ISCAS*, pages 1–5. IEEE, 2018.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941. ACM/SIAM, 2000.

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[KRR+18] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Constant-time discrete Gaussian sampling. *IEEE Trans. Computers*, 67(11):1561–1571, 2018.

[KRVV19] Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on falcon. *IACR Cryptology ePrint Archive*, 2019:267, 2019.

[KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[MAR17] Carlos Aguilar Melchor, Martin R. Albrecht, and Thomas Ricosset. Sampling from arbitrary centered discrete Gaussians for lattice-based cryptography. In *ACNS*, volume 10355 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2017.

[MBdD+10] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[MKMS21] Jose Maria Bermudo Mera, Angshuman Karmakar, Tilen Marc, and Azam Soleimanian. Efficient lattice-based inner-product functional encryption. *IACR Cryptol. ePrint Arch.*, page 46, 2021.

[MLL+13] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X. . Zhou, and J. L. O'Brien. Experimental realisation of Shor's quantum factoring algorithm using qubit recycling. In *2013 Conference on Lasers Electro-Optics Europe International Quantum Electronics Conference CLEO EUROPE/IQEC*, pages 1–1, May 2013.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012.

[MR18] Carlos Aguilar Melchor and Thomas Ricosset. CDT-based Gaussian sampling: From multi to double precision. *IEEE Trans. Computers*, 67(11):1610–1621, 2018.

[MSO17] Sarah McCarthy, Neil Smyth, and Elizabeth O'Sullivan. A practical implementation of identity-based encryption over NTRU lattices. In *IMACC*, volume 10655 of *Lecture Notes in Computer Science*, pages 227–246. Springer, 2017.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In *CRYPTO (2)*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, 2017.

[MWG+21] Soundes Marzougui, Nils Wisiol, Patrick Gersch, Juliane Krämer, and Jean-Pierre Seifert. Machine-learning side-channel attacks on the GALACTICS constant-time implementation of BLISS. *CoRR*, abs/2109.09461, 2021.

[NIS15] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. https://doi.org/10.6028/NIST.FIPS.202, 2015.

[NIS16a] NIST. NIST post-quantum competition. http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf, 2016. Accessed: 2018-10-31.

[NIS16b] NIST. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. https://doi.org/10.6028/NIST.SP.800-185, 2016.

[OSHG19] Tobias Oder, Julian Speith, Kira Höltgen, and Tim Güneysu. Towards practical microcontroller implementation of the signature scheme falcon. In *PQCrypto*, volume 11505 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2019.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In *CCS*, pages 1843–1855. ACM, 2017.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.

[Pei10] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2010.

[Pes16] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *INDOCRYPT*, volume 10095 of *Lecture Notes in Computer Science*, pages 153–170, 2016.

[PFH+17] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. https://falcon-sign.info/, 2017. Accessed: 2018-10-31.

[POG15] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In *LATINCRYPT*, volume 9230 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2015.

[Por19] Thomas Pornin. New efficient, constant-time implementations of falcon. *IACR Cryptol. ePrint Arch.*, page 893, 2019.

[PP19] Thomas Pornin and Thomas Prest. More efficient algorithms for the NTRU key generation using the field norm. In *Public Key Cryptography (2)*, volume 11443 of *Lecture Notes in Computer Science*, pages 504–533. Springer, 2019.

[Pre15] Thomas Prest. Gaussian sampling in lattice-based cryptography, 2015.

[Pre17] Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 347–374. Springer, 2017.

[PRR19] Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Simple, fast and constant-time Gaussian sampling over the integers for Falcon. Second PQC Standardization Conference, https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/rossi-simple-fast-constant.pdf, 2019. Accessed: 2019-08-13.

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *DATE*, pages 1697–1702. IEEE, 2017.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

[RRVV14] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Compact and side channel secure discrete Gaussian sampling. *IACR Cryptology ePrint Archive*, 2014:591, 2014.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RVV13] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. High precision discrete Gaussian sampling on fpgas. In *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 383–401. Springer, 2013.

[Saa16] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for ring-lwe. *IACR Cryptology ePrint Archive*, 2016:276, 2016.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography. *IACR Cryptology ePrint Archive*, 2018:39, 2018.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2011.

[SSZ18] Ron Steinfeld, Amin Sakzad, and Raymond K. Zhao. Titanium: Proposal for a nist post-quantum public-key encryption and kem standard specifications document version 1.1. http://users.monash.edu.au/~rste/Titanium_v11.pdf, 2018. Submitted to NIST Post-Quantum Competition. Accessed: 2019-01-08.

[SZJ+21] Shuo Sun, Yongbin Zhou, Yunfeng Ji, Rui Zhang, and Yang Tao. Generic, efficient and isochronous gaussian sampling over the integers. *IACR Cryptol. ePrint Arch.*, 2021:199, 2021.

[TLLV07] David B. Thomas, Wayne Luk, Philip Heng Wai Leong, and John D. Villasenor. Gaussian random number generators. *ACM Comput. Surv.*, 39(4):11, 2007.

[von51] John von Neumann. Various techniques used in connection with random digits. In A.S. Householder, G.E. Forsythe, and H.H. Germond, editors, *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 1951.

[ZCHW17] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, and William Whyte. NIST PQ submission: pqNTRUSign a modular lattice signature scheme. https://www.onboardsecurity.com/nist-post-quantum-crypto-submission, 2017. Accessed: 2019-08-01.

[ZMS+21] Raymond K. Zhao, Sarah McCarthy, Ron Steinfeld, Amin Sakzad, and Máire O'Neill. Quantum-safe HIBE: does it cost a latte? *IACR Cryptol. ePrint Arch.*, page 222, 2021.