Accurate and Composable Noise Estimates for CKKS with Application to Exact HE Computation

Jean-Philippe Bossuat[1,4], Anamaria Costache[2], Christian Mouchet[3], Lea Nürnberger[2] and Juan Ramón Troncoso-Pastoriza[4]

[1] Gauss Labs Pte. Ltd., Singapore
[2] Norwegian University of Science and Technology – NTNU, Information Security and Communication Technology, Trondheim, Norway
[3] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
[4] Tune Insight SA, Lausanne, Switzerland

Abstract. All RLWE-based FHE schemes are inherently noisy. The CKKS scheme (Cheon, Kim, Kim, Song, Asiacrypt 2017) considers the noise as a part of the message, yielding approximate computations but also considerable performance gains. Since it grows with each homomorphic operation and incurs a precision loss, it is paramount for users to be able to estimate the noise level throughout a given circuit in order to appropriately estimate parameters and control the precision loss in the message. In this work, we develop a noise model that allows for tight estimates of the precision loss, and propose a tool prototype for computing these estimates on any given circuit. Our noise model relies on a novel definition, the component-wise noise, which makes the average-case noise estimates tighter and more composable. As a result, our model and tool can derive accurate estimates of complex circuits such as bootstrapping. We experimentally demonstrate the tightness of our noise estimates by showing that our theoretical estimates never deviate by more than 0.01 bits from experimental estimates, even for large circuits, and hold with high probability. Furthermore, we demonstrate how to apply our techniques to obtain an exact version of the CKKS scheme in which the decryption removes all the noise (with high probability). Such a scheme has many applications, as it allows to take advantage of the efficiency of CKKS, while preserving an exact message space, hence further strengthening CKKS against IND-CPA-D attacks.

Keywords: Fully Homomorphic Encryption · CKKS · Noise Analysis · Exact CKKS

1 Introduction

Fully Homomorphic Encryption (FHE) enables meaningful computations on encrypted data. Following Gentry's initial construction from standard lattice assumptions, current constructions [BGV12, CKKS17, CGGI20, DM15, FV12, GSW13] are based on variants of the Learning-with-Error problem (LWE) [Reg09]; improving their efficiency has been a significant endeavor of the field. The cornerstone of efficiency is to manage *noise*: a small random deviation added for security purposes during the encryption process which grows throughout the computation and which must be filtered out during decryption. Hence, the correctness of current FHE schemes depends on successful message decoding and is therefore probabilistic.

E-mail: jeanphilippe.bossuat@gmail.com (Jean-Philippe Bossuat), anamaria.costache@ntnu.no (Anamaria Costache), christian.mouchet@hpi.de (Christian Mouchet), lea.nurnberger@ntnu.no (Lea Nürnberger), juan@tuneinsight.com (Juan Ramón Troncoso-Pastoriza)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2025-08-11.

One of the most popular schemes to date for computing on approximate real and complex numbers was introduced by Cheon et al. [CKKS17] and is dubbed CKKS. Its efficiency stems from a simple noise management strategy: it considers the error as part of the approximation error of its underlying plaintext, resulting in an *approximate* scheme. That is, instead of returning the plaintext $\mathtt{pt}$, the decryption algorithm returns $\mathtt{pt}+e$, where e is a small noise term – no attempts are made to separate the noise from the plaintext. This is different in so-called *exact* FHE schemes ([BGV12, CGGI20, DM15, FV12, GSW13]). These apply more advanced encoding strategies to make sure that the noise can be completely removed during decryption: BGV [BGV12] scales the noise e with the plaintext modulus t. This leads to the noise always being a multiple of the plaintext modulus, as long as it does not wrap around the ciphertext modulus. Thus, a reduction modulo the plaintext modulus completely removes the noise, and the decryption algorithm only returns the plaintext $\mathtt{pt}$. In BFV, the message is scaled so that the noise is encoded in the lower bits, allowing it to be rounded off as long as it remains below a certain bound. These techniques enable exact decryption with a high probability, but result in a performance loss compared to CKKS. In CKKS, the noise e is considered part of the plaintext and is not removed. Thus, CKKS suffers from what is called precision loss: the more operations are performed on ciphertexts, the more the noise e will grow, slowly polluting the least significant bits of the message, and reducing the number of message bits that are returned exactly. Moreover, it has been shown that an adversary with access to the raw decryption error through decryption can mount passive key-recovery attacks [LM21]. Such attacks are captured by the so-called IND-CPA-D model, and all countermeasures known to date require the user to compute a bound on the decryption error. Therefore, for both correctness and security reasons, it is essential for users to quantify this precision loss/noise growth throughout a computation.

1.1 Our Contributions

Our contributions in this work are as follows. First, we present noise estimates for all homomorphic operations in CKKS. Based on these estimates, we propose a tool, dubbed the estimator, that can estimate the noise incurred by the evaluation of any circuit in CKKS before runtime. Secondly, we make progress towards achieving IND-CPA-D security for CKKS. We show how to use our noise estimates to obtain an exact version of the CKKS scheme with some high probability δ. We call this scheme δ-exact CKKS.

In more detail, our contributions are as follows.

- **Composable Average-Case Noise Bounds:** We extend the existing work on average-case noise analysis by (1) proposing a novel *component-wise noise* approach to track the noise and (2) supporting plaintext-ciphertext operations, rotations and conjugations. Thanks to these extensions, we can derive bounds on complex circuits such as iterative algorithms or bootstrapping, which were left unaddressed by previous works. We additionally present noise estimates for all the operations needed in bootstrapping, such as homomorphic matrix-ciphertext products, encoding and decoding. We experimentally confirm that our bounds are tight: they do not deviate from the experimentally observed noise by more than 0.01 bits in all experiments.

- **Open-Source Average-Case Noise Estimator:** We provide a proof-of-concept open-source estimator[1] that calculates these noise bounds for any circuit. To do so, the estimator *simulates* the noise at every step of the circuit. For input gates, this is done by drawing fresh values according to the (user-provided) plaintext distribution. For the arithmetic gates, this is done by sampling the noise distribution, according to our composable bounds. Note that this can be done at circuit design time, i.e.,

[1]The code is available at https://github.com/tuneinsight/ckks-noise-estimator

before the actual plaintext values are known (as long as their distribution is). This contribution is of independent interest: it provides a crucial development tool to assess how FHE circuits behave and to fine-tune parameters in order to achieve a target precision for a given application. Our estimator targets (and is implemented in) the Lattigo library [Lat24].

- **Exact CKKS:** We construct a δ-exact scheme that returns the message exactly with probability δ. We achieve this by rounding off the noise after decryption, based on probabilistic noise bounds, and by showing how the condition of "correctability" set forth in [CCH+22] can be achieved. We validate our construction experimentally. This constitutes a major step towards IND-CPA-D security of CKKS, since perfectly exact schemes are trivially secure in this model. However, we could not prove IND-CPA-D security for this version of CKKS. This is because our noise estimates rely on two heuristics (commonly used in the literature [CCH+23, CNP23, BMCM23, MP24]: (1) that all the noise terms are independent, and (2) that they are distributed according to a Gaussian distribution. Indeed, while the noise distribution is well approximated by a Gaussian (we verify this experimentally), we cannot formally quantify "how close" it is to one, which would be required to prove full IND-CPA-D security.

1.2 Related Work

1.2.1 Noise analysis for CKKS

Costache et al. [CCH+23] provided the first average-case noise analysis for CKKS. They developed noise estimates for encoding, encryption, addition, multiplication, relinearization, and rescaling for both CKKS and RNS-CKKS [CHK+19]. We go further than the noise estimates given in [CCH+23], as we also perform noise analysis for addition and multiplication by plaintexts and constants, rotations, conjugations, and key-switchings in general. We use a different approach, tracking the component-wise noise (Definition 2), instead of the precision loss directly. Furthermore, we demonstrate that our noise estimates can be combined into estimates for deep circuits such as bootstrapping and provide experimental validation.

1.2.2 "Exact" CKKS

In the full version of their paper [CCH+22], Costache et al. also formalize the approach of exact CKKS by rounding off the noise bits in the element $\mathtt{pt} + e$ output by the decryption algorithm. However, they do not fully achieve an exact CKKS scheme, since they assume circuits to be "correctable" in their proof; how circuits can be made correctable was left as an open problem. We extend their work by showing how to obtain correctable circuits.

A different and recent topic of interest has been how to remove a certain part of the noise in CKKS. The motivation for this approach is that the noise growth in CKKS means that only circuits of limited depth can be meaningfully evaluated, since the noise eventually will swallow the message (unless the circuit benefits from self-correcting properties). Drucker et al. [DMPS24] propose a method to reduce the error of a CKKS ciphertext encrypting binary values via the homomorphic evaluation of a step function. This keeps the error small enough such that circuits of arbitrary depth, over this plaintext space, can be evaluated. But this work seeks a different goal than ours: [DMPS24] focuses on the ability to evaluate circuits of unlimited depth over binary values, whereas our work is focused on achieving exactness over rationals of some fixed precision.

Bae et al. [BCKS24] build on the work by Drucker et al. [DMPS24] and propose new bootstrapping algorithms that reduce the noise in CKKS ciphertexts encrypting binary values, similar to bootstrapping in schemes such as BGV, BFV, and TFHE. Two follow-up papers [BKSS25, AKP24] address some of the remaining open problems, including the

extension of this technique to larger integer values. The direction of these papers is different than ours: they focus on controlling the growth of the noise during evaluation. Conversely, we do not aim at reducing the noise during evaluation, but focus on enabling correct decryption.

1.2.3 On average-case noise analysis

A recent work [GZ25] calls the average-case approach into question. In this paper, the authors make a valid critique on how common heuristic assumptions are necessary for average-case analyses (such as ours) to function. In more detail, the authors investigate the validity of common heuristics, including those that we rely on: that the noise terms are independent and follow a Gaussian distribution. Although the paper demonstrates how these heuristics easily fail in corner cases, it also points out that such failure cases did not lead to any decryption failure *in practice* for their case study (i.e., the OpenFHE parameter selection tool). Hence, the conclusion from [GZ25] does not differ much from ours: while such heuristics are not suitable to argue about the security of a scheme (this prevents us from proving IND-CPA-D security for our version of exact CKKS), they still enable precise noise analysis and meaningful applications in circuit design and parameter selection.

1.2.4 Comparison with related work

We present a short comparison between our bounds and the bounds obtained in [CCH+23]. We find that for fresh ciphertexts, addition, and rescaling our bounds agree with the bounds presented in [CCH+23]. However, while the bounds for multiplication presented in [CCH+23] underestimate the experimentally observed noise by as much as 5 bits, our bounds are tight (we observe a deviation from the theory of at most 0.01 bits. Additionally, we consider many complex operations that [CCH+23] do not, and we also test our bounds on large circuits (particularly, on deep circuit). Our bounds remain tight for large circuits, which is not expected for [CCH+23], since their multiplication bounds underestimate the experimentally observed noise.

Drucker et al. [DMPS24] also introduce an exact CKKS scheme. Their techniques and targeted functionality, however, are very different from ours. Their approach aims at providing SIMD gate bit-wise arithmetic, which they achieve by homomorphically applying a step function after a number of gates have been evaluated, in order to maintain the noise below some threshold. Recent works have extended the framework of Drucker et al. by providing exact arithmetic over small integers [BCKS24, BKSS25, AKP24, Kim25], but similarly to Drucker et al., the plaintext space remains small (e.g. 8 bits). Overall, these works aim to provide the functionality of single ciphertext schemes (such as TFHE), but with the efficiency and throughput of SIMD schemes. In contrast, we target SIMD exact fixed point complex arithmetic over correctable circuits, and we achieve this by providing very tight noise bounds for the given homomorphic circuit that is evaluated, which allows us to remove the noise upon decryption. Finally, we note that our noise bounds have applications beyond changing CKKS to be exact – for example, for determining appropriate levels of noise-flooding for IND-CPA-D security [BCDS+25].

2 Preliminaries

2.1 Algebraic Background

For a positive integer m, let $\Phi_m(X)$ be the m-th cyclotomic polynomial of degree $N = \phi(m)$, where ϕ is the Euler totient function. We will assume m to be a power of two. Then, we have $N = \frac{m}{2}$, and $\Phi_m(X) = X^N + 1$. Let $\mathcal{R} := \mathbb{Z}[X]/(\Phi_m(X))$ be the ring of integers of

a number field $\mathbb{Q}[X]/(\Phi_m(X))$. We denote by $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ the residue ring of $\mathcal{R}$ modulo an integer Q. Elements of $\mathcal{R}_Q$ are polynomials of degree lower than N. We refer to $\mathcal{R}_Q$ as the plaintext space and to $(\mathcal{R}_Q)^k, k > 1$ as the ciphertext space. We will mostly identify an element $p \in \mathcal{R}_Q$ by its coefficient vector. Any norm $||\cdot||$ on an element from $\mathcal{R}_Q$ is to be understood as a norm on its coefficient vector.

For CKKS, we define a chain of ciphertext moduli $Q_L > Q_{L-1} > \ldots > Q_0$, where $Q_i|Q_j \ \forall i \leq j$. Let $q_0, \ldots, q_L$ be machine-word sized NTT-friendly primes and choose Q_ℓ such that $Q_\ell = \prod_{j=0}^{\ell} q_j$. Unless otherwise stated, we will consider an element from $\mathcal{R}_{Q_\ell}$ in its unique RNS representation $\mathcal{R}_{q_0} \times \ldots \times \mathcal{R}_{q_\ell} \cong \mathcal{R}_{Q_\ell}$.

2.2 Notation

We denote a vector $\mathbf{v}$ with boldface letter. For a polynomial f, we denote its i^{th} coefficient by f_i and $\lceil f \rfloor$ refers to coefficient-wise rounding to the nearest integer. Let p be a polynomial randomly drawn from $\mathcal{R}_{Q_\ell}$. Then we denote the standard deviation of the i-th coefficient by σ_{p_i}. When applied to a vector $\mathbf{P} = (p^{(1)}, \ldots, p^{(k)})$ of random polynomials, this notation applies component-wise: $\sigma_{\mathbf{P}_i} = (\sigma_{p_i^{(1)}}, \ldots, \sigma_{p_i^{(k)}})$ is the vector of standard deviations for the i-th coefficients of each component of $\mathbf{P}$. If it is clear that all the coefficients of p are identically distributed, we drop the index i.

2.3 Products of Random Polynomials and Error Function

We now present several definitions and results on which our analysis relies. We use the term "random polynomial" to refer to a polynomial for which the coefficients are identically and independently drawn from a distribution $\mathcal{D}$. The coefficient standard deviation of these polynomials is then the standard deviation of $\mathcal{D}$. We use the following result from [CNP23].

Lemma 1 (Products of Random Polynomials [CNP23])**.** *Let $f, g \in \mathcal{R}$ be two polynomials of degree N, for which the coefficients are drawn identically and independently from two distributions $\mathcal{D}_f$ and $\mathcal{D}_g$, with variances σ_f^2 and σ_g^2, and means μ_f and μ_g, respectively. Then, the i-th coefficient of $f \cdot g$ has variance of $\sigma^2_{(fg)_i}$ with:*

$$\sigma^2_{(fg)_i} = \mu_f^2\sigma_g^2 + \mu_g^2\sigma_f^2 + N\sigma_g^2\sigma_f^2.$$

In the case that $f = g$ we need to specialise the above result as follows. We furthermore specialise to symmetric distributions centered around zero, since we will only need the following lemma in this context.

Lemma 2 (Squaring of Random Polynomials)**.** *Let $f \in \mathcal{R}$ be a polynomial of even degree N, with coefficients drawn identically and independently from a symmetric distribution $\mathcal{D}$ with standard deviation σ_f and expectation 0. Then:*

$$\begin{cases} \sigma_{(f^2)_i} = \sigma_f^2\sqrt{2N+1} & \text{, for } i \text{ even} \\ \sigma_{(f^2)_i} = \sigma_f^2\sqrt{2N} & \text{, for } i \text{ odd.} \end{cases}$$

For large N, we can therefore approximate the standard deviation of $(f^2)_i$ for all i by $\sqrt{2N}\sigma_f^2$.

Proof. See Appendix A. □

We will furthermore need the following definition and well-known lemmas for bounding the noise distributions.

Definition 1 (Error Function)**.** The error function $\texttt{erf} : \mathbb{R} \to (-1, 1)$ is the function defined as

$$\texttt{erf}(z) = \frac{2}{\pi} \int_0^z e^{-t^2} dt.$$

Lemma 3 (Bounding Gaussian Distributions)**.** *Let $X \sim \mathcal{N}(0, \sigma^2)$ be a normally distributed random variable with standard deviation σ. Then we have*

$$\mathsf{Pr}(-\alpha\sigma \leq X \leq \alpha\sigma) = \textit{erf}\left(\frac{\alpha}{\sqrt{2}}\right).$$

For example, we have $1 - \texttt{erf}\left(\frac{6}{\sqrt{2}}\right) \approx 2^{-30}$, $1 - \texttt{erf}\left(\frac{10}{\sqrt{2}}\right) \approx 2^{-76}$, and $1 - \texttt{erf}\left(\frac{14}{\sqrt{2}}\right) \approx 2^{-145.5}$ as the probability that $|X|$ is larger than 6, 10 and 14 standard deviations, respectively.

2.4 RLWE-based Encryption

The secret-key of the CKKS scheme is a polynomial $s \in \mathcal{R}_Q$ and we write $\texttt{sk} = (1, s, s^2, \ldots, s^k)$. The ciphertexts are tuples in $\mathcal{R}_Q^{k+1}$ with $k \geq 1$ and we will write them as $\texttt{ct} = (\texttt{ct}[0], \texttt{ct}[1], \ldots, \texttt{ct}[k])$. The scheme encrypts vectors of complex values $\mathbf{m} \in \mathbb{C}^n$, $n \leq N/2$ into $\texttt{ct} \in \mathcal{R}_Q^{k+1}$ such that

$$\texttt{Decode}([\langle \texttt{ct}, \texttt{sk} \rangle]_Q) = \texttt{Decode}\left(\left[\sum_{i=0}^{k} \texttt{ct}[i] \cdot s^i\right]_Q\right) \approx \mathbf{m},$$

for some decoding function `Decode` that we describe in the next section. Note that for fresh ciphertexts and after most operations $k = 1$. We will therefore assume ciphertexts of length 2 unless explicitly stated otherwise.

2.5 Messages Encoding and Plaintext Space

The encoding of a message $\mathbf{m} \in \mathbb{C}^n$ is performed by the $\texttt{Encode} : \mathbb{C}^n \to \mathcal{R}_{Q_\ell}$ algorithm via the inverse canonical map $\tau^{-1} : \mathbb{C}^n \to \mathbb{R}[Y]/(Y^{2n} + 1)$ for $Y = X^{N/2n}$. We refer to n as the the number of *plaintext slots*. We will call elements in the ring $\mathcal{R}_{Q_\ell}$ *plaintexts*, and elements in $\mathbb{C}^n$ *messages*.

Through scaling and rounding, the element $\tau^{-1}(\mathbf{m})$ is discretized onto an element $\texttt{pt}$ in the ring $\mathcal{R}_{Q_\ell}$; τ^{-1} is computed as follows: (1) Compute $\mathbf{m}' = \texttt{DFT}_n^{-1}(\mathbf{m})$ the inverse Discrete Fourier Transform (DFT) of $\mathbf{m}$. (2) Extract the real and imaginary parts of $\mathbf{m}'$ as $\mathbf{m}'_0 := \frac{1}{2}(\mathbf{m}' + \overline{\mathbf{m}'}) = \mathrm{Re}(\mathbf{m}')$ and $\mathbf{m}'_1 := \frac{-i}{2}(\mathbf{m}' - \overline{\mathbf{m}'}) = \mathrm{Im}(\mathbf{m}')$ and set $\mathbf{m}'_0 || \mathbf{m}'_1 \in \mathbb{R}^{2n}$. Then $\tau^{-1}(\mathbf{m})$ is a polynomial in $\mathbb{R}[Y]/(Y^{2n} + 1)$ for $Y = X^{N/2n}$ with coefficients $\mathbf{m}'_0 || \mathbf{m}'_1$. Thus, the first n coefficients of $\tau^{-1}(\mathbf{m})$ are the real parts of $\texttt{DFT}_n^{-1}(\mathbf{m})$ and the second n coefficients, the imaginary part of $\texttt{DFT}_n^{-1}(\mathbf{m})$. (3) Transform $\tau^{-1}(\mathbf{m})$ into an element in $\mathcal{R}_{Q_\ell}$ by calculating $\texttt{pt} = \lceil \Delta \cdot \tau^{-1}(\mathbf{m}) \rfloor$ for a scaling factor Δ, and apply the change of variable $Y \to X^{N/2n}$. `Decode` is the inverse operation.

Remark 1. The structure of the polynomial $\tau^{-1}(\mathbf{m}) \in \mathbb{R}[Y]/(Y^{2n} + 1)$ is $\tau^{-1}(\mathrm{Re}(\mathbf{m})) + \tau^{-1}(\mathrm{Im}(\mathbf{m})) \cdot Y^n$, were $\tau^{-1}(\mathrm{Re}(\mathbf{m}))$ and $\tau^{-1}(\mathrm{Im}(\mathbf{m}))$ are polynomials of the subring $\mathbb{R}[Y + Y^{-1}]/(Y^{2n} + 1) \subset \mathbb{R}[Y]/(Y^{2n} + 1) \subseteq \mathbb{R}[X]/(X^N + 1)$.

2.6 Noise Definitions

Noise analysis in CKKS comes with two extra challenges: encoding into the plaintext domain is only approximately correct, and ciphertexts have a scaling factor that can change

during the evaluation of a circuit. Both the approximate encoding and the scale-change introduce noise. For a ciphertext $\mathtt{ct}$ encrypting m, the decryption does not return m but a noisy term $m + e$ that differs from the original message. This difference e is what we consider as noise. For $\mathtt{ct}$ encrypting $\mathtt{pt}$, and $\mathtt{pt}$ the encoding of m, the noise can be calculated as $[\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q - \mathtt{pt}$ (before decoding) and as $\mathtt{Decode}([\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q) - m$ (after decoding). For instance, [CCH+23] tracks the precision loss through a computation in CKKS by tracking the variance of $[\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q - \mathtt{pt}$.

2.7 Average-case and Worst-case Noise Analysis

Two different approaches to track the noise in FHE schemes have been proposed: worst-case noise analysis and average-case noise analysis. In a worst-case analysis, the noise in the ciphertext is *bounded* at encryption and for each step of the computation. These bounds are derived by assuming that the random variable falls within a certain number of standard deviations (this number is selected to be $\alpha = 6$ for example in [CS16]). For example, let $\mathtt{ct}_0$ and $\mathtt{ct}_1$ be two ciphertexts. The worst-case approach then consists in determining (possibly probabilistic) bounds B_0 and B_1 on $\mathtt{ct}_0$ and $\mathtt{ct}_1$ respectively. Let $\epsilon_i = B_i - ||\langle \mathtt{ct}_i, \mathtt{sk} \rangle||_\infty, i \in \{0, 1\}$, the amount by which the bound deviates from the real noise. The worst-case bound after an addition between $\mathtt{ct}_0$ and $\mathtt{ct}_1$ can then be calculated from B_0 and B_1 as $B = B_0 + B_1$. This new bound deviates then by $\epsilon = \epsilon_0 + \epsilon_1$ from the real noise (i.e., the looseness has doubled). While such bounds are very safe, they are also very loose in practice [CLP20], because the bounding error grows with the circuit depth. Since loose bounds lead to an overly conservative choice of parameters, which in turn have an impact on efficiency, we are interested in obtaining as tight bounds as possible.

The average-case analysis follows a different approach: instead of deriving a bound at each step, we track the variance of the noise through every homomorphic computation, and only derive a probabilistic bound at the very end. For example, to bound the noise of $\mathtt{ct} = \mathtt{ct}_0 + \mathtt{ct}_1$ we first compute the standard deviation $\sigma_{\mathtt{ct}}$ from $\sigma_{\mathtt{ct}_0}$ and $\sigma_{\mathtt{ct}_1}$ (assuming that the noise is normally distributed). Only then, we use an appropriate tail bound on the output distribution. This leads to much more precise error bounds (whose approximation error does not grow with the circuit) and the ability to tightly control the failure probability of the bounds. The authors of [CCH+23] showed that the noise after each operation in CKKS can be well approximated by a Gaussian distribution in each component, assuming independent inputs. While theoretically assuming independence is a stretch, this assumption can be experimentally validated (see for example [CNP23]). Therefore, the infinity norm of a complex vector of length n can be shown to lie with probability $\delta = \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$ in an interval of α standard deviations around 0 (we consider a complex vector as a real vector of twice the size).

In this work, we follow the average-case approach. In Section 3, we calculate the standard deviation of the noise in all basic operations in RNS-CKKS. These results can then be combined to obtain the coefficient standard deviation of the noise after the evaluation of an arbitrary circuit. From this standard deviation, a bound on the precision loss can be obtained as follows.

Lemma 4. *Let $\mathtt{ct}$ be a CKKS ciphertext with component-wise noise $n(\mathtt{ct})$ and component-wise noise coefficient standard deviation $\sigma_{n(\mathtt{ct})} = (\sigma_{n(\mathtt{ct})[0]}, \sigma_{n(\mathtt{ct})[1]}, \dots)$ encrypting a plaintext $\mathtt{pt}$ that encodes a complex vector of length $\frac{N}{2}$. Let $\mathtt{sk} = (1, s, s^2, \dots, s^k)$ be the secret key, and σ_{s^k} the standard deviation of the $k+1$st component of the secret-key. Then we have*

$$||\mathtt{Decode}(\langle \mathtt{ct}, \mathtt{sk} \rangle - \mathtt{pt})||_\infty \leq \alpha \sqrt{\sigma^2_{n(\mathtt{ct})[0]} + \sum_{i>0} N \sigma^2_{n(\mathtt{ct})[i]} \sigma^2_{s^i}},$$

with probability $\delta = \texttt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^N, \alpha \in \mathbb{R}$.

3 Average-case Noise Estimates for RNS-CKKS

This section introduces our noise estimates, with the following structure: Section 3.1 presents our component-wise noise definition, Sections 3.2 to 3.5 present noise estimates for individual operations. These noise estimates are composable: for each basic HE operation, we provide formulas describing the distribution of the output noise as a function of the input noise. Then, we can simply compose the estimates for these basic operations into estimates for general circuits. We provide examples for concrete circuits such as matrix multiplication and bootstrapping in Appendix B. The noise estimates form the core of our proposed automated noise estimator, that we present in Section 4.

Since the estimator is implemented with and for the Lattigo library [Lat24], we fine-tune our noise estimates for Lattigo's RNS-CKKS implementation. Due to space reasons, we cannot give all derivations and proofs in this section. We fully include those that are sufficiently short, and only sketch the proofs that best illustrate our general proof techniques. The remaining proofs can be found in Appendix A. In this section, we will use the following notation: Let $\mathtt{pt}^{(0)}, \mathtt{pt}^{(1)}$, be two plaintexts encoding $\mathbf{m}_0, \mathbf{m}_1 \in \mathbb{C}^n$. Let Δ_0 be the scaling factor of $\mathtt{pt}^{(0)}$ and let Δ_1 be the scaling factor of $\mathtt{pt}^{(1)}$. Let $n(\mathtt{pt}^{(0)}), n(\mathtt{pt}^{(1)})$ denote the noise (i.e., the rounding error) from encoding the messages $\mathbf{m}_0, \mathbf{m}_1$ into $\mathtt{pt}_0, \mathtt{pt}_1$ respectively. Let $\mathtt{ct}^{(0)}, \mathtt{ct}^{(1)}$ be the ciphertexts encrypting $\mathtt{pt}^{(0)} + n(\mathtt{pt}^{(0)}), \mathtt{pt}^{(1)} + n(\mathtt{pt}^{(1)})$ with respect to a ciphertext modulus Q_{ℓ_0}, Q_{ℓ_1} respectively. Let $\mathtt{pt} \in \mathcal{R}_Q$ be a plaintext with scaling factor Δ, and with standard deviation of encoding noise $\sigma_{n(\mathtt{pt})_i}$. Whenever a plaintext-ciphertext operation is performed, we assume that the plaintext modulus is the same as the ciphertext modulus and that its scaling factor is chosen optimally (e.g. to the be same in the case of addition or the next prime to be removed by the rescaling in the case of multiplication).

3.1 Component-wise Noise

We will follow a novel approach and track the noise in each ciphertext component $\mathtt{ct}[i], i \in [0, k]$ (i.e., "before decryption"). This quantity can then be combined in a coefficient standard deviation of the precision loss.

Definition 2 (Component-wise Noise)**.** Let $\mathtt{ct}$ be a degree-k CKKS encryption of a plaintext $\mathtt{pt} \in \mathcal{R}_{Q_\ell}$, $k > 0$. Then, $\mathtt{ct} = (\mathtt{pt} + a_0 + e_0, a_1 + e_1, \ldots, a_k + e_k)$ for some $a_0, \ldots, a_k \in \mathcal{R}_{Q_\ell}$ such that $a_0 = -\sum_{i=1}^{d} a_i s^i$, and we define the *component-wise noise* of the ciphertext $\mathtt{ct}$ as $n(\mathtt{ct}) = (e_0, e_1, \ldots, e_k)$.

In other words, the component-wise noise captures all the terms introduced during encryption and homomorphic evaluation which do not get canceled out during decryption. Note that this includes the error introduced by encoding, where relevant. By using this definition, we can accurately estimate the noise growth of operations for which the output noise does not depend on all components. For instance, the key-switching operation does not depend on $n(\mathtt{ct})[0]$ and accurately modeling its output noise requires distinguishing between the different noise components. Then, the actual precision loss can be obtained from the component-wise noise:

$$\langle \mathtt{ct}, \mathtt{sk} \rangle - \mathtt{pt} = \sum n(\mathtt{ct})[i] \cdot s^i.$$

3.2 Noise Estimates for Plaintext Encoding

Let $\mathbf{m} \in \mathbb{C}^n$ be a message. Since DFT_n and DFT_n^{-1} and therefore $\tau(\cdot)$ and $\tau^{-1}(\cdot)$ are inverses of one another, we have $\tau(\tau^{-1}(\mathbf{m})) = \mathbf{m}$. However, since $\mathtt{Encode}(\mathbf{m})$ additionally

scales and rounds the result, we have instead $\texttt{Decode}(\texttt{Encode}(\mathbf{m})) \approx \mathbf{m}$. The difference between $\texttt{Decode}(\texttt{Encode}(\mathbf{m}))$ and $\mathbf{m}$ is given by the rounding error. The rounding error is deterministic; however, it depends on the precise operations that have been performed on the message. As is standard, we therefore model it as uniformly random in the interval $\left[-\frac{1}{2}, \frac{1}{2}\right)$, with variance $\frac{1}{12}$ [CCH+23]. We denote the encoding error by $n(\texttt{pt})$, that is, we have $\texttt{pt} + n(\texttt{pt}) = \lceil \Delta\tau^{-1}(\mathbf{m}) \rfloor$.

3.3 Noise Estimates for Encryption and Rescaling

We begin by giving the noise estimates for rescaling, since they will be needed in all subsequent noise estimates, including encryption.

Lemma 5 (Rescaling)**.** *Let* $\{\texttt{ct}, Q_\ell, \Delta\}$ *be a ciphertext and let* $\texttt{ct}^{(1)} = \texttt{DropLevel}(\{\texttt{ct}, Q_\ell, \Delta\}, k)$ *and* $\texttt{ct}^{(2)} = \texttt{Rescale}(\{\texttt{ct}, Q_\ell, \Delta\})$*, both encrypting* $\texttt{pt}$*. Then we have for the standard deviation of the component-wise noise*

$$\sigma_{n(\texttt{ct}^{(1)})} = \sigma_{n(\texttt{ct})} \quad \textit{and} \quad \sigma_{n(\texttt{ct}^{(2)})} = \left(\sqrt{q_\ell^{-2}\sigma^2_{(n(\texttt{ct})[0])} + \frac{1}{12}}, \sqrt{q_\ell^{-2}\sigma^2_{(n(\texttt{ct})[1])} + \frac{1}{12}}\right).$$

In particular, if $||\sigma_{(n(\texttt{ct})[i])})||_\infty \leq \frac{q_\ell}{2}$*, then* $\sigma_{(n(\texttt{ct}^{(2)})[i]))} \approx \sqrt{\frac{1}{12}}$*, for* $i \in \{0, 1\}$*.*

Proof. During $\texttt{Drop Level}$ the ciphertext $\texttt{ct}^{(1)}$ is simply considered with respect to a different ciphertext modulus, without being scaled. The noise therefore remains the same. $\texttt{ct}^{(2)}$ is calculated as follows.

$$\texttt{ct}^{(2)} = \lceil q_\ell^{-1}\texttt{ct} \rfloor = q_\ell^{-1}\texttt{ct} + (\tau^{(0)}, \tau^{(1)}),$$

where the rounding errors $\tau^{(i)}$ are polynomials with coefficients in $\left(-\frac{1}{2}, \frac{1}{2}\right]$. Since their coefficients can be assumed to be continuously uniformly distributed in that interval, they have coefficient standard deviation $\sigma_{\tau^{(i)}} = \sqrt{\frac{1}{12}}$.

If $\texttt{ct}$ has some noise $n(\texttt{ct})$ it is merely scaled by q_ℓ^{-1}, and some rounding noise $(\tau^{(0)}, \tau^{(1)})$ is added. Thus we have for the component-wise noise after rescaling $n(\texttt{ct}^{(2)}) = q_\ell^{-1}n(\texttt{ct}) + (\tau^{(0)}, \tau^{(1)})$. Then, the coefficient standard deviations of $n(\texttt{ct}^{(2)})$ is

$$\sigma_{n(\texttt{ct}^{(2)})} = \left(\sqrt{q_\ell^{-2}\sigma^2_{(n(\texttt{ct})[0])} + \frac{1}{12}}, \sqrt{q_\ell^{-2}\sigma^2_{(n(\texttt{ct})[1])} + \frac{1}{12}}\right).$$

□

Remark 2. Rescaling is called to make up for the exponential growth of the scaling factor Δ. Therefore, ideally the rescaling of Δ^2 should be done by a factor Δ^{-1} instead of q_ℓ^{-1}. However, because we can only divide the coefficients by a factor of the modulus, which is Q_ℓ in the RNS-CKKS, the updated scaling factor is not $\Delta^2/\Delta = \Delta$ but Δ^2/q_ℓ. However, by keeping track of the exact scaling factor and ensuring that additions are done between ciphertexts of the same scaling factor, we can avoid the introduction of any new error. Additionally, it is possible to write any circuit such that the scaling factor always eventually falls back to Δ by appropriately scaling its inputs.

We proceed by giving the noise estimates for fresh public-key ciphertexts and plaintexts.

Lemma 6 (Encoding and Fresh Encryption Noise)**.** *Let* $\mathbf{m}$ *be a message, and let* $\texttt{pt} \in \mathcal{R}_{Q_\ell}$ *be the encoding of* $\mathbf{m}$ *with scaling factor* Δ*. Let* $\texttt{ct}_{pk}$ *be a public-key RNS-CKKS ciphertext, and let* $\texttt{ct}_{sk}$ *be a secret key RNS-CKKS ciphertext, both encrypting* $\texttt{pt}$ *using the respective* $\texttt{Encrypt}()$ *function. Let* σ_0 *be the standard deviation of the error distribution.*

Then we have for the coefficient standard deviation of component-wise noise

$$\sigma_{n(\mathtt{pt})} = \sqrt{\frac{1}{12}} \qquad \sigma_{n(\mathtt{ct}_{pk})} = \left(\sqrt{\frac{1}{6}}, \sqrt{\frac{1}{12}}\right) \qquad \sigma_{n(\mathtt{ct}_{sk})} = \left(\sqrt{\sigma_0^2}, 0\right).$$

Proof. For reasons of space, the full proof can be found in Appendix A. We sketch it here:

- The noise in the plaintext stems from the rounding after encoding. The rounding noise can be modelled as a continous uniform random variable over the interval $\left[-\frac{1}{2}, \frac{1}{2}\right)$. Thus it has variance $\frac{1}{12}$.
- A public-key ciphertext is created as follows: first, an encryption of 0 is created with respect to a ciphertext modulus QP. Here, there are two sources of randomness: the randomness from the encryption, and the randomness of the public key. Then, we apply a scaling by P^{-1} and a rounding operation. This introduces a rounding noise. Since the encryption and public-key noise are small, the scaling and rounding by P^{-1} operation makes them negligible compared to the rounding noise. Finally, the plaintext is added. The final noise of a fresh encryption therefore consists of the plaintext and the rounding noise in the first component, and only the rounding noise in the second component.
- A secret-key ciphertext is created by simply drawing $a \xleftarrow{\$} \mathcal{R}_{Q_L}$ and $e \leftarrow \chi$, and computing $(as + e, a)$. Thus, the noise of a secret-key ciphertext consists of the encryption randomness in the first component, and is zero in the second.

□

3.4 Noise Estimates for Additions and Tensor Products

We next give the noise estimates for additions, additions by constants and message vectors, as well as for tensor products, and multiplications by constants and message vectors. The difference between constants and message vectors lies in the shape of their respective encoded plaintext: encoded constants output monomials while encoded vectors output full polynomials. In what follows we will refer to encoded message vectors simply as plaintexts.

Lemma 7 (Addition by Plaintext)**.** *Let* $\{\mathtt{ct}^{(2)}, Q_{\ell_0}, \Delta_0\} = \mathtt{AddPlain}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{pt} + n(\mathtt{pt}), Q_{\ell_0}, \Delta_0\})$ *be the result of a plaintext-ciphertext addition. Then we have for the standard deviation of the component-wise noise of* $\mathtt{ct}^{(2)}$*, and for the standard deviation of a plaintext-plaintext addition*

$$\sigma_{n(\mathtt{pt}^{(0)}+\mathtt{pt}^{(1)}))} = \sqrt{\sigma^2_{n(\mathtt{pt}^{(0)})} + \sigma^2_{n(\mathtt{pt}^{(1)})}} \quad \textit{and}$$
$$\sigma_{n(\mathtt{ct}^{(2)})} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \sigma^2_{n(\mathtt{pt})}}, \sigma_{n(\mathtt{ct}^{(0)})[1]}\right).$$

Proof. Adding two plaintexts results in their noise being added. The result follows.

When adding a plaintext to a ciphertext, we can set the plaintext's scale to be the same as the ciphertext's, which is known. Therefore, we can consider the plaintext and the ciphertext to be at the same scale, and there is no need for scale-adaptation. A ciphertext $\mathtt{ct}^{(0)}$ consists in the first component of the plaintext $\mathtt{pt}^{(0)}$, some mask $a^{(00)}$ that gets cancelled out during decryption and the noise $n(\mathtt{ct}^{(0)})[0]$. Similarly, the second component also consists of some mask $a^{(01)}$ and the noise $n(\mathtt{ct}^{(0)})[1]$. Adding those two gives

$$\mathtt{ct}^{(2)} = (\mathtt{pt}^{(0)} + \mathtt{pt} + (a^{(00)} + n(\mathtt{ct}^{(0)})[0]) + n(\mathtt{pt}), (a^{(01)} + n(\mathtt{ct}^{(0)})[1])).$$

Since the masks are cancelled out during decryption, and we do not consider the plaintexts as part of the component-wise noise, from the above it can easily be seen that the component-wise noise of $\mathtt{ct}^{(2)}$ is given by $n(\mathtt{ct}^{(2)}) = (n(\mathtt{ct}^{(0)})[0] + n(\mathtt{pt}), n(\mathtt{ct}^{(0)})[1])$. Thus we obtain for the coefficient standard deviation of $n(\mathtt{ct}^{(2)})$:

$$\sigma_{n(\mathtt{ct}^{(2)})} = (\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \sigma^2_{n(\mathtt{pt})}}, \sigma_{n(\mathtt{ct}^{(0)})[1]}).$$

□

In CKKS, the first to $\frac{N}{2}$-th coefficients of a ciphertext polynomial encode the real part of the plaintext, and the $\frac{N}{2}+1$st to N-th coefficients encode the imaginary part. If a plaintext is added to a ciphertext, all these coefficients will be changed. However, if a constant (a plaintext of the form $a + bX^{\frac{N}{2}}$) is added to a ciphertext, only the first and $\frac{N}{2}+1$st coefficient are changed. Thus, the ciphertext coefficients no longer are identically distributed. Thus, we need to give separate formulas for the standard deviation of the first and $\frac{N}{2}+1$st coefficient, and the remaining coefficients. We added an index i to the vector of standard deviations, to mark this difference.

Lemma 8 (Addition by Constant)**.** *Let* $\mathtt{const} \in \mathbb{C}^n$ *be a constant. Let* $\{\mathtt{ct}^{(2)}, Q_{\ell_0}, \Delta_0\} = \mathtt{AddConst}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \mathsf{const})$ *the result of a constant-ciphertext addition. Then we have for the standard deviation of the component-wise noise*

$$\sigma_{n(\mathtt{ct}^{(2)})_{i=0,N/2}} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})_{i=0,N/2}[0]} + \frac{1}{12}}, \sigma_{n(\mathtt{ct}^{(0)})_{i=0,N/2}[1]}\right)$$
$$\sigma_{n(\mathtt{ct}^{(2)})_{i\neq 0,N/2}} = \left(\sigma_{n(\mathtt{ct}^{(0)})_{i\neq 0,N/2}[0]}, \sigma_{n(\mathtt{ct}^{(0)})_{i\neq 0,N/2}[1]}\right).$$

Proof. $\mathtt{ct}^{(2)}$ is calculated by directly adding a constant. A constant is a monomial, and therefore its real part only gets added to the first coefficient encoding the real parts ($i = 0$), and its imaginary part gets added to the first coefficient encoding the imaginary part ($i = \frac{N}{2}$). The constant gets scaled and rounded, therefore the rounding noise gets added along with the constant. Since the constant only gets added to the first ciphertext component, the noise of $\mathtt{ct}^{(3)}[0]$ consists of the noise of $\mathtt{ct}^{(0)}[0]$ and the rounding noise, whereas $n(\mathtt{ct}^{(3)}[1]) = n(\mathtt{ct}^{(0)}[1])$. □

Lemma 9 (Addition by Ciphertext)**.** *Let* $\{\mathtt{ct}^{(2)}, \min(Q_{\ell_0}, Q_{\ell_1}), \max(\Delta_0, \Delta_1)\} = \mathtt{Add}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{ct}^{(1)}, Q_{\ell_1}, \Delta_1\})$ *be the result of a ciphertext-ciphertext addition. Then we have for the standard deviation of the component-wise noise*

$$\sigma_{n(\mathtt{ct}^{(2)})} = \left(\sqrt{\left(\frac{\max(\Delta_0,\Delta_1)}{\Delta_0}\right)^2 \sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \left(\frac{\max(\Delta_0,\Delta_1)}{\Delta_1}\right)^2 \sigma^2_{n(\mathtt{ct}^{(1)})[0]}},\right.$$
$$\left.\sqrt{\left(\frac{\max(\Delta_0,\Delta_1)}{\Delta_0}\right)^2 \sigma^2_{n(\mathtt{ct}^{(0)})[1]} + \left(\frac{\max(\Delta_0,\Delta_1)}{\Delta_1}\right)^2 \sigma^2_{n(\mathtt{ct}^{(1)})[1]}}\right).$$

Proof. The two ciphertexts $\mathtt{ct}^{(0)}$ and $\mathtt{ct}^{(1)}$ do not necessarily have the same scale Δ. Therefore, the first step is to align their scales. This is done by multiplying $\mathtt{ct}^{(0)}$ with $\frac{\max(\Delta_0,\Delta_1)}{\Delta_0}$, and $\mathtt{ct}^{(1)}$ by $\frac{\max \Delta_0,\Delta_1}{\Delta_1}$ respectively. We assume that this fraction returns an integer, thus we do not need to round after the multiplication, and therefore do not get a rounding error. Then the lemma follows: the component-wise noise that is already present in the ciphertexts gets multiplied by the scaling factor and then added. The statement of the lemma follows. □

During a multiplication, the ciphertext noise is multiplied by both the plaintext and the plaintext noise: the encoding and the ciphertext noise are intertwined and both contribute to the component-wise noise $n(\mathtt{ct})$. Therefore, from now on, we no longer track the encoding noise separately. Since the plaintext impacts the noise, it becomes necessary to discuss how to model it. There are two possible approaches for modeling the plaintexts. The first is to assume that we are modeling a specific application, therefore the plaintexts are known. In that case, we do not model them as random variables, but rather as fixed values. The second approach is to assume that we are modeling a general circuit, and not restrict the model to specific plaintext values. In this case, we treat the plaintext values as random variables, following a distribution with standard deviation $\sigma_{\mathtt{pt}}$. To make our results precise, the distribution will have to be taken over a bounded interval; in our case, we assume the interval $[-1,1]^n \cup [-i,i]^n \in \mathbb{C}^n$ for the messages. We now give the noise estimates for the more general case and model the plaintexts as random variables.

Lemma 10 (Multiplication by Plaintext)**.** *Let* $\mathtt{pt}_\times = (\mathtt{pt}^{(0)} + n(\mathtt{pt}^{(0)}))(\mathtt{pt}^{(1)} + n(\mathtt{pt}^{(1)})$ *be the plaintext product, and* $\{\mathtt{ct}^{(2)}, Q_{\ell_0}, \Delta_0 q_{\ell_0}\} = \mathtt{MultPlain}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{pt} + n(\mathtt{pt}), Q, q_{\ell_0}\}))$*. Then we have:*

$$\sigma_{n(\mathtt{pt}_\times)} = \sqrt{N\left(\sigma^2_{n(\mathtt{pt}^{(0)})}\sigma^2_{\mathtt{pt}^{(1)}} + \sigma^2_{n(\mathtt{pt}^{(1)})}\sigma^2_{\mathtt{pt}^{(0)}} + \sigma^2_{n(\mathtt{pt}^{(0)})}\sigma^2_{n(\mathtt{pt}^{(1)})}\right)}$$
$$\sigma_{n(\mathtt{ct}^{(2)})} = \left(\sqrt{N\sigma^2_{n(\mathtt{ct}^{(1)})[0]}\left(\sigma^2_{\mathtt{pt}} + \sigma^2_{n(\mathtt{pt})}\right)}, \sqrt{N\sigma^2_{n(\mathtt{ct}^{(1)})[1]}\left(\sigma^2_{\mathtt{pt}} + \sigma^2_{n(pt)}\right)}\right).$$

Proof. We sketch the proof as follow: A plaintext consists of the exact value and the approximation error. Thus, the noise of a plaintext-plaintext product is given by $(\mathtt{pt}^{(0)} + n(\mathtt{pt}^{(0)})(\mathtt{pt}^{(1)} + n(\mathtt{pt}^{(1)})) - \mathtt{pt}^{(0)}\mathtt{pt}^{(1)}$. The expression for the standard deviation follows. In a plaintext-ciphertext product, the ciphertext noise gets multiplied with the plaintext and the plaintext noise with the ciphertext and both become a part of the resulting overall ciphertext noise. The result follows. □

Lemma 11 (Multiplication by constant)**.** *Let* $\mathtt{const} = a + bi \in \mathbb{C}^n$ *be a constant. Let* $\{\mathtt{ct}^{(2)}, Q_{\ell_0}, \Delta_0 q_{\ell_0}\} = \mathtt{MultConst}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \mathtt{const}, q_{\ell_0})$*. Then we have for the coefficient standard deviation*

$$\sigma_{n(\mathtt{ct}^{(2)})} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\left((a^2+b^2)q_{\ell_0} + \frac{1}{6}\right)}, \sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left((a^2+b^2)q_{\ell_0} + \frac{1}{6}\right)}\right).$$

Proof. The product with a complex constant is computed as follows. The ciphertext $\mathtt{ct}^{(0)}$ gets scaled by both aq_{ℓ_0} and bq_{ℓ_0} separately. The product $\lceil bq_{\ell_0}\mathtt{ct}^{(0)}\rfloor$ is rotated by $\frac{N}{2}$ coefficients through multiplying it with $X^{\frac{N}{2}}$. Thus $\lceil bq_{\ell_0}\mathtt{ct}^{(0)}\rfloor X^{\frac{N}{2}}$ now encrypts $b\mathrm{Im}(c_i)$ in the first $\frac{N}{2}$ coefficients and $b\mathrm{Re}(c_i)$ in the second half. However, this rotation does not introduce any extra noise, since it merely permutes the coefficients. The component-wise noise is therefore scaled by $(a+b)q_{\ell_0}$ and two times the rounding noise gets added. We therefore obtain the following result

$$n(\mathtt{ct}^{(2)})_{i=0,\frac{N}{2}}[j] = n(\mathtt{ct}^{(0)})_{i=0,\frac{N}{2}}[j]aq_{\ell_0} + \tau^{(0)} + n(\mathtt{ct}^{(0)})_{i=0,\frac{N}{2}}[j]bq_{\ell_0} + \tau^{(1)}$$
$$n(\mathtt{ct}^{(2)})_{i=0,\frac{N}{2}}[j] = n(\mathtt{ct}^{(0)})_{i=0,\frac{N}{2}}[j]bq_{\ell_0} + \tau^{(2)} + n(\mathtt{ct}^{(0)})_{i=0,\frac{N}{2}}[j]aq_{\ell_0} + \tau^{(3)},$$

with $j \in \{0,1\}$ and $\tau^{(k)}$ the rounding polynomials. Since we can assume them to have coefficients continuously uniformly distributed in $\left(-\frac{1}{2}, \frac{1}{2}\right]$, they all have standard deviation

$\frac{1}{12}$. Thus we get the claimed results for the standard deviations.

$$\sigma_{n(\mathtt{ct}^{(2)})} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\left((a^2+b^2)q_{\ell_0}+\frac{1}{6}\right)}, \sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left((a^2+b^2)q_{\ell_0}+\frac{1}{6}\right)}\right).$$

□

Lemma 12 (Tensor Product)**.** *Assume* $\mathtt{ct}^{(0)} \neq \mathtt{ct}^{(1)}$. *Let* $\{\mathtt{ct}^{(2)}, \min(Q_{\ell_0}, Q_{\ell_1}), \Delta_0\Delta_1\} = \mathtt{Tensor}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{ct}^{(1)}, Q_{\ell_1}, \Delta_1\})$, *and let* $\{\mathtt{ct}^{(3)}, Q_{\ell_0}, \Delta_0^2\} = \mathtt{Tensor}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\})$, *the result of a squaring. Then we have for the coefficient standard deviation of the component-wise noise*

$$\begin{aligned}
&\sigma_{n(\mathtt{ct}^{(2)})} \\
&= \left(\sqrt{N\left(\sigma^2_{pt^{(0)}}\sigma^2_{n(\mathtt{ct}^{(1)})[0]} + \sigma^2_{pt^{(1)}}\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{n(\mathtt{ct}^{(1)})[0]}\right)},\right. \\
&\quad \sqrt{N\left(\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left(\sigma^2_{pt^{(1)}} + \sigma^2_{n(\mathtt{ct}^{(1)})[1]}\right) + \sigma^2_{n(\mathtt{ct}^{(1)})[1]}\left(\sigma^2_{pt^{(0)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[1]}\right)\right)}, \\
&\quad \left.\sqrt{N}\sigma_{n(\mathtt{ct}^{(0)})[1]}\sigma_{n(\mathtt{ct}^{(1)})[1]}\right).
\end{aligned}$$

$$\begin{aligned}
&\sigma^2_{n(\mathtt{ct}^{(3)})} \\
&= \left(\sqrt{2N\left(2\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{pt^{(0)}} + \sigma^4_{n(\mathtt{ct}^{(0)})[0]}\right)}, \sqrt{4N\left(\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left(\sigma^2_{pt^{(0)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[0]}\right)\right)},\right. \\
&\quad \left.\sqrt{2N}\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\right).
\end{aligned}$$

Proof. For reasons of space, we present a sketch of the proof. The full proof can be found in Appendix A.

The tensor product is calculated as $\mathtt{ct}_\otimes = (\mathtt{ct}_0[0]\mathtt{ct}_1[0], \mathtt{ct}_0[1]\mathtt{ct}_1[0] + \mathtt{ct}_0[0]\mathtt{ct}_1[1], \mathtt{ct}_0[1]\mathtt{ct}_1[1])$. The noise of $\mathtt{ct}_b[0]$, $b \in \{0,1\}$, consists of the ciphertext noise (rounding noise, encryption randomness, key randomness), and the plaintext noise. The noise in the second component $\mathtt{ct}_b[1]$ consists in only the ciphertext noise. Thus, the noise in the first component of the tensor product is a product of those noise terms $n(\mathtt{ct}_\otimes)[0] = (n(\mathtt{ct}_0)[0] + n(\mathtt{pt}_0))(n(\mathtt{ct}_1)[0] + n(\mathtt{pt}_1)) = n(\mathtt{ct}_0)[0]n(\mathtt{ct}_1)[0] + n(\mathtt{ct}_0)[0]n(\mathtt{pt}_1) + n(\mathtt{pt}_0)n(\mathtt{ct}_1)[0] + n(\mathtt{pt}_0)n(\mathtt{pt}_1)$. The noise components $n(\mathtt{ct}_\otimes)[1]$ and $n(\mathtt{ct}_\otimes)[2]$ can be calculated similarly. The standard deviation of the noise components can be obtained by repeatedly applying Lemma 1 and 6. □

3.5 Noise Estimates for Key Switching, Relinearization, Rotation, and Conjugation

We now proceed to give a proof for the noise estimates after key switching. Our noise estimates are based on the optimised version of the evaluation keys as introduced by [BMTPH21].

Lemma 13 (Key-Switch)**.** *Let* $\mathtt{ct}'$ *be a ciphertext and* $\mathtt{ct} = \mathtt{ct}' + \mathtt{KeySwitch}(\mathtt{ct}'[i], s)$ *for some secret* s*. Then we have for the component-wise noise of* $\mathtt{ct}$

$$n(\mathtt{ct}) = n(\mathtt{ct}') + \left(n(\mathtt{ct}'[i])s + P^{-1} \left(\sum_{j=0}^{\beta-1} [\mathtt{ct}'[i]]_{q_{\gamma_j}} n(\mathtt{evk}[0]) \right) + \tau^{(0)}, \tau^{(1)} \right),$$

and for the coefficient standard deviation of $n(\mathtt{ct})$

$$\sigma_{n(\mathtt{ct})} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}'[0])} + N\sigma^2_{n(\mathtt{ct}'[i])}\sigma_s^2 + \frac{N \sum_{j=0}^{\beta-1} q_{\gamma_j}^2 \sigma_0^2}{12P^2} + \frac{1}{12}}, \sqrt{\sigma^2_{n(\mathtt{ct}'[1])} + \frac{1}{12}} \right).$$

To shorten notation later we will denote

$$n_{\mathsf{ks-add}}(\mathtt{ct}[i], s) = \left(n(\mathtt{ct}[i])s + P^{-1} \left(\sum_{j=0}^{\beta-1} [\mathtt{ct}[i]]_{q_{\gamma_i}} n(\mathtt{evk}[0]) \right) + \tau^{(0)}, \tau^{(1)} \right),$$

and

$$\sigma_{\mathsf{ks-add}}(\mathtt{ct}[i], s) = \left(\sqrt{N\sigma^2_{n(\mathtt{ct})[i]}\sigma_s^2 + \frac{N \sum_{i=0}^{\beta-1} q_{\gamma_i}^2 \sigma_0^2}{12P^2} + \frac{1}{12}}, \sqrt{\frac{1}{12}} \right),$$

the component-wise noise and the coefficient standard deviation of the component-wise noise added to the first component of a ciphertext while key switching.

Proof. For reasons of space, we present a sketch of the proof. The full proof can be found in Appendix A.

During key switching, the (three-element) ciphertext is first scaled to a modulus QP. The third component is decomposed and multiplied with the relevant evaluation key. The result is scaled by P^{-1}, rounded and added to the first ciphertext component. The second ciphertext component is simply scaled and rounded. Therefore, the noise in the first component consists of the scaled and rounded sum and product from the key-switching procedure. Additionally, we have the rounding noise from the scaling operation, as well as the previous noise from the first ciphertext component. In the second component, we simply have the rounding noise. The standard deviations can be obtained by repeatedly applying Lemma 1, and by modelling the decomposition of the ciphertext as discrete uniform random variables over $\mathcal{R}_{Q_i}$. □

Lemma 14 (Relinearization)**.** *Let* $\{\mathtt{ct}, Q_\ell, \Delta\}$ *be a ciphertext resulting from* `Tensor`*. Note that* $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1], \mathtt{ct}[2])$*. Let* $\{\mathtt{ct}, Q_\ell, \Delta\} = \mathtt{Relin}(\{\mathtt{ct}, Q_\ell, \Delta_0\}) = (\mathtt{ct}[0], \mathtt{ct}[1]) + \mathtt{KeySwitch}(\mathtt{ct}[2], s^2)$*.*

$$\sigma_{n(\mathtt{Relin}(\mathtt{ct}))} = \sqrt{\left(\sigma^2_{n(\mathtt{ct}[0])}, \sigma^2_{n(\mathtt{ct}[1])}\right) + n_{\mathsf{ks-add}}(\mathtt{ct}[2], s^2)}.$$

Proof. The proof can be trivially derived from the proof of Lemma 13. □

We base our noise estimates for the rotation on an improved version of the rotation algorithm as proposed in Algorithm 4 in [BMTPH21], where the rotation keys for a rotation by k slots are given as $\mathtt{evk}_{s \to \Phi_{k^{-1}}(s)}$. The algorithm is stated in Appendix C.

Lemma 15 (Rotation)**.** *Let* $\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}$ *be a ciphertext encrypting* $\mathtt{pt}^{(0)}$ *with respect to a secret key* $\mathtt{sk} = (1, s)$*. Let* $\{\mathtt{ct}, Q_{\ell_0}, \Delta_0\} = \mathtt{Rotate}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, k)$ *be a ciphertext encrypting* $\Phi_k(m_{ct})$ *with respect to* $\Phi_k(\mathtt{sk}) = (1, \Phi_{k^{-1}}(s))$*. Then we have for the coefficient standard deviation of the component-wise noise*

$$\sigma_{rot(ct)} = \sqrt{\sigma^2_{n(ct)} + \sigma^2_{\mathsf{ks-add}(ct[1], \Phi_{k^{-1}}(s))}} \ .$$

Proof. Since Φ is a permutation that induces a sign change at most, the proof can be trivially derived from the proof of Lemma 13. □

Lemma 16 (Conjugation)**.** *Let* $\mathtt{ct}^{(0)}$ *be a ciphertext encrypting* $\mathtt{pt}^{(0)}$*, which is an encoding of* m_0 *and let* $\mathtt{ct} = \mathtt{Conjugation}(\mathtt{ct}^{(0)})$ *be the ciphertext encrypting* $\mathtt{pt}$*, encoding* $\overline{m_0}$*. Then we have for the coefficient standard deviation of the component-wise noise*

$$\sigma_{conj(ct)} = \sqrt{\sigma^2_{n(ct)} + \sigma^2_{\mathsf{ks-add}(ct[1], \Phi^{-1}(s))}} \ .$$

Proof. Since Φ is a permutation that at most induces a sign change, the proof can be trivially derived from the proof of Lemma 13. □

4 Estimator

We implement a functional noise estimator whose goals are twofold: (i) to validate that our theoretical noise estimations of Section 3 are correct and that they hold for a wide range of large depth circuits with high accuracy and (ii) to experimentally, and independently of the theoretical analysis, find the minimum amount of information that is necessary to accurately track the noise.

As a result, and unlike previous noise estimations approaches, our estimator does not track the variance of the message and noise through a single variable. Such an approach works in the average case for basic operations, but it breaks down in practice when we start to consider structured plaintexts or start to compose basic operations. Instead, and similarly to how noise estimates are derived in Section 3, we experimentally concluded that for an estimator to be accurate, it needs to track each and every coefficient of the message and the noise components. The estimator therefore not only models how the noise evolves during homomorphic operations, but also what the expected plaintext result would be. This enables users to verify the correctness and behavior of an encrypted circuit, without having to perform computations in the encrypted domain.

4.1 Internal Workings

The estimator models its homomorphic operations, and how they are performed, on the implementation provided by the Lattigo library [Lat24]. It supports all the elementary homomorphic operations, as well as some of the advanced ones, such as linear transformations, homomorphic DFTs, polynomial evaluation and bootstrapping.

Unlike previous approaches, which operate on messages characterized by their distribution, our estimator requires the user to provide concrete messages. The estimator then adds the expected encoding and encryption noise and operates on the message as it would be done under encryption. Then, it updates the values throughout the user-defined circuit. The message and noise are stored in the Canonical embedding for two reasons: (i) arithmetic can be carried out as point-wise arithmetic over complex numbers and (ii) we do not need to evaluate DFTs when multiplying elements. This approach enables all linear noise propagation to be automatically and efficiently carried out, without requiring approximations (e.g. addition or tensoring). The noise approximation is only introduced

when non-linear operations are called, such as the rescaling or key-switching, to ensure that they remain efficient. During these operations, only the part that would modify the plaintext is exact; the rest, for example how the noise is expected to evolve, is approximated. We perform these approximations by sampling fresh noise according to the derivations of Section 3 and performing an approximate mapping to the Canonical embedding by scaling it by $1/\sqrt{N}$ (which avoids having to call a DFT).

4.2 Performance

The current implementation of our estimator should be viewed as a proof of concept that validates our approach, and not as a final product. It does not benefit from an optimized implementation and in its current state it uses arbitrary precision complex arithmetic to ensure that all computations are accurate, bringing it on par with the time necessary to run the computation in the encrypted domain. We however believe that an optimized implementation could significantly improve its efficiency, namely because most operations in the estimator are Hadamard products or additions, and the most time-consuming operation under encryption, the key-switching, is replaced in the estimator by a noise sampling. Regardless, running the computation with the estimator requires a much lower memory than running the computation in the encrypted domain since no evaluation keys are needed and computations do not require to be carried out modulo a large prime.

4.3 Experiments

We perform several experiments to assess the precision of our estimator. These experiments are described in Paragraphs I to VIII below and their results are summarized in Table 1. For each experiment, we report `AVG` and `STD`, the average and standard deviation of the precision across all slots respectively, for the estimator (`Est`) and encrypted values (`Enc`). The precision is computed as the negative base-2 logarithm of the L2-norm of the error in the canonical embedding, i.e. $-\log_2(\sqrt{e\bar{e}})$ (this enables to have a single statistic for both the real and imaginary values). It can be interpreted as the number of matching bits after the decimal. All experiments are carried out in a ring degree of $N = 2^{16}$. To further validate our estimator, we conduct each experiment with varying scaling factors of 2^{45} and 2^{55}, and a varying secret-key Hamming-weight: $2N/3$ and 192. We should expect an increase of ≈ 10 bits of precision when switching from the scaling factor 2^{45} to the scaling factor 2^{55}, as well as an increase in precision of $\log_2(\sqrt{2N/3}) - \log_2(\sqrt{192}) \approx 3.9$ bits when switching from the secret with Hamming-weight $2N/3$ to the secret with Hamming-weight 192. All statistics were computed from data gathered over 128 runs (i.e. $2^{15} \cdot 2^7 = 2^{22}$ slots).

I. High Degree Chebyshev Power. In this experiment we evaluate $T_{4096}(\mathtt{ct})$, i.e. the 2^{12}-th Chebyshev power of the first kind, on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-1, 1]$. The total circuit depth is 12. The results of this experiment can be found in Set I of Table 1.

II. High Degree Chebyshev Approximation of the Sigmoid. In this experiment we evaluate a Chebyshev approximation of $1/(e^{-x} + 1)$ in the interval $[-32, 32]$, on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-31, 31]$. The total circuit depth is 8. The results of this experiment can be found in Set II of Table 1.

III. Goldschmidt Division. In this experiment we compute an approximate inverse using the Goldschmidt division algorithm with 10 iterations on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-0.1, 1.9]$. The total circuit depth is 10. The results of this experiment can be found in Set III of Table 1.

Table 1: Results for the experiments described in Section 4.3. All experiments use ring degree $N = 2^{16}$, $\log(Q)$ is the base two logarithm of the ciphertext modulus, $\log(P)$ the base two logarithm of the auxiliary prime (used during the key-switching), h the Hamming weight of the secret, L the multiplicative depth enabled by the parameters and $\log(\Delta)$ the base two logarithm of the scaling factor. `AVG` and `STD` are the average and standard deviation of the bit-precision predicted by the estimator (`Est`) and empirically verified (`Enc`).

Set	$\log(Q)$	$\log(P)$	h	L	$\log(\Delta)$	AVG		STD	
						Est	Enc	Est	Enc
I	595	183	$2N/3$	12	45	20.19	20.19	2.48	2.48
			192			24.07	24.06	2.46	2.45
	720	183	$2N/3$		55	30.19	30.19	2.48	2.48
			192			34.07	34.07	2.45	2.45
II	460	183	$2N/3$	8	45	30.89	30.88	2.01	2.01
			192			34.79	34.79	1.99	1.99
	500	183	$2N/3$		55	40.88	40.88	2.01	2.02
			192			44.78	44.78	1.99	1.99
III	505	183	$2N/3$	10	45	29.54	29.55	2.08	2.08
			192			33.43	33.44	2.05	2.06
	550	183	$2N/3$		55	39.54	39.54	2.08	2.08
			192			43.44	43.43	2.06	2.05
IV	685	305	$2N/3$	14	45	30.89	30.89	2.01	2.01
			192			35.08	35.08	1.88	1.88
	830	305	$2N/3$		55	41.19	41.19	1.90	1.90
			192			45.09	45.09	1.88	1.89
V	240	183	$2N/3$	4	45	31.20	31.18	1.80	1.79
			192			35.09	35.08	1.79	1.78
	280	183	$2N/3$		55	41.19	41.18	1.80	1.79
			192			45.09	45.08	1.79	1.78
VI	240	183	$2N/3$	4	45	25.96	25.92	1.62	1.60
			192			28.15	28.15	1.60	1.60
	280	183	$2N/3$		55	35.96	35.93	1.62	1.60
			192			38.15	38.15	1.60	1.60
VII	588	244	$2N/3$	9	45	32.32	32.32	1.86	1.86
			192			36.09	36.06	2.01	1.99
VIII	920	244	$2N/3$	16	45	23.68	23.67	1.61	1.60
			192			27.04	27.41	1.60	1.60

IV. Binary Values Cleaning. In this experiment we clean binary values having an error of ± 0.2 by 7 sequential evaluations of the polynomial $-2x^3 + 3x^2$ on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-0.2, 0.2] \cup [0.8, 1.2]$, according to the method proposed in [DMPS24]. The total circuit depth is 14. Note that 7 sequential evaluations are more that what would be needed to reach the maximum expected precision, however this enables to test how our estimator behaves with iterated circuits that converge toward discrete values. The results of this experiment can be found in Set IV of Table 1.

V. CoeffsToSlots. In this experiment we homomorphically evaluate the map $\tau^{-1}(\cdot)$ on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-1, 1] \cup [-i, i]$. This circuit can be regarded as evaluating a complex inverse DFT of dimension 2^{15}. The inverse DFT matrix is factorized into 4 sparse complex matrices (depth 4 circuit) and is configured to return the real and imaginary part into distinct ciphertexts (similarly to what is done in the bootstrapping). The results can be found in Set V of Table 1.

VI. SlotsToCoeffs. In this experiment we homomorphically evaluate the map $\tau^{(\cdot)}$ on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-1, 1] \cup [-i, i]$. This circuit can be regarded as evaluating a complex DFT of dimension 2^{15}. The DFT matrix is factorized into 4 sparse complex matrices (depth 4 circuit). The results of this experiment can be found in Set VI of Table 1.

VII. EvalMod. In this experiment we homomorphically evaluate the reduction modulo Q/Δ on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-1, 1]$, on which multiples $k \cdot Q/\Delta$ for $k \in [-K + 1, K - 1]$ and $K = 16$ were added. The homomorphic modular reduction is done by evaluating $(Q/(2\pi\Delta)) \sin(2\pi x\Delta/Q)$, which approximated with a degree 30 Chebyshev interpolant of the phase-shifted scaled cosine function, followed by 3 evaluations of the double angle formula, for a total depth of 9 (one being used for the change of basis to the Chebyshev polynomial). This circuit was not run with a scaling factor of 2^{55} for two reasons: (i) because it requires the initial message ratio $Q_0/|m|$ to be at least 2^8 and the maximum prime size allowed by Lattigo is 61 and (ii) the scaling factor actually used during the circuit is 2^{60} regardless of the initial scaling factor. The results of this experiment can be found in Set VII of Table 1.

VIII. Bootstrapping In this experiment we evaluate the bootstrapping circuit on a ciphertext encrypting a vector of 2^{15} values uniformly distributed in the interval $[-1, 1] \cup [-i, i]$. We used the default bootstrapping parameters of the Lattigo library which notably uses an ephemeral secret of $\tilde{h} = 32$ to provide a failure probability of $2^{-138.7}$ and an expected average precision of 23.9-bits for $h = N/2$ (since we use $h = 2N/3$ and compute the precision as the L2-norm, we expect a slightly lower precisoin) and 27.4 for $h = 192$ [BTPH22]. The depth of the default bootstrapping circuit is 15. Note that estimating the noise of the bootstrapping requires more than composing the homomorphic step of the bootstrapping (`CoeffsToSlots`, `EvalMod`, `SlotsToCoeffs`), as it also requires to carefully match each low-level parameterization that the Lattigo library would perform and that would have a direct or indirect impact on the final result but also sample the polynomial $I(X)$ according to the expected distribution. This circuit was not run with a scaling factor of 2^{55} for three reasons: (i) because it requires the initial message ratio $Q_0/|m|$ to be at least 2^8 and the maximum prime size allowed by Lattigo is 61, (ii) the scaling factor actually used during the EvalMod step is 2^{60} regardless of the initial scaling factor and (iii) it would not affect the final precision since the error added during the bootstrapping circuit is independent of the initial scaling factor. The results can be found in Set VIII of Table 1.

5 Exact CKKS

In this section, we apply our results from Section 3 to present an exact version of the CKKS scheme. An exact version of the CKKS scheme can have many different applications. First, it allows to use CKKS for applications requiring exact results while preserving the advantages of CKKS. Second, it can be a step towards achieving IND-CPA-D security. We discuss this latter application in more detail in Section 5.4. We begin by defining what we mean by an *exact* FHE scheme.

Definition 3 ((δ, r)-exact FHE scheme)**.** Let $\mathcal{E} = (\mathsf{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ be a (levelled) FHE scheme. We say that $\mathcal{E}_{\delta,r} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}_{\delta,r}, \mathsf{Eval})$ is a (δ, r)-exact scheme, if for any valid ciphertext $\mathtt{ct}$ encrypting a message $m = \sum_{i=0}^{N} m_i 2^i$ we have that

$$\mathsf{Dec}_{\delta,r}(\mathtt{ct}, \mathtt{sk}) = \sum_{i=r}^{N} m_i 2^i,$$

with probability δ.

We make some remarks about the above definition.

- If $\delta = 1$ and $r = 0$, then $\mathcal{E}_{1,0}$ is what can be called a perfectly exact scheme – note that none of the known FHE schemes fulfil this definition.
- If $1 > \delta \gg 0$ and $r = 0$, then $\mathcal{E}_{\delta,0}$ is what in FHE is commonly called an exact scheme: a scheme that returns the message exactly, with high probability, unless a decryption failure occurs.
- If $1 > \delta \gg 0$ and $r > 0$ then $\mathcal{E}_{\delta,r}$ corresponds to a scheme where only the most significant r bits of the message are recovered.

In CKKS the noise is not separated from the plaintext. Thus, the decryption result will always be of the form $\mathtt{pt} + e$, for $||e||_\infty < ||\mathtt{pt}||_\infty$ a noise term. The most straightforward attempt at achieving exactness for CKKS would therefore be to remove the lowest $r = \lceil \log_2(||e||_\infty) \rceil$ bits of the decryption result, and thus to obtain a (δ, r)-exact scheme. This can be achieved as follows: for $\Delta' = 2^{r+1}$, the error term can removed by multiplying the decryption result with $\frac{1}{\Delta'}$. Then, rounding should remove the error. That is, we could define a decryption functionality for CKKS that results in a (δ, r)-exact scheme by defining

$$\mathsf{Dec}^{\mathsf{naive}}_{\delta,r}(\mathtt{ct}, \Delta', \mathtt{sk}) = \mathsf{round}\left(\frac{1}{\Delta'}\mathtt{Dec}(\mathtt{ct}, sk)\right) = \mathsf{round}\left(\frac{1}{\Delta'}(\mathtt{pt} + e)\right).$$

Here, round can be either flooring, ceiling or rounding. This, however, does not work. The reasons are as follows.

- Consider $\mathsf{round}(\cdot) = \lceil \cdot \rfloor$. Then, we require that $||(\frac{1}{\Delta'}\mathtt{pt} - \lceil \frac{1}{\Delta'}\mathtt{pt} \rfloor) + (\frac{1}{\Delta'}e - \lceil \frac{1}{\Delta'}e \rfloor)||_\infty \leq \frac{1}{2}$. The bounds from Section 3 allow us to get a bound on e. Thus, by choosing $\Delta' = \frac{1}{2^{B+1}}$, where $B > \lceil \log_2(||e||_\infty) \rfloor$ is a bound on the noise that holds with probability δ, we have that $\lceil \frac{1}{\Delta'}e \rfloor = 0$ and we have that

$$\left|\left|\frac{1}{\Delta'}e - \left\lceil \frac{1}{\Delta'}e \right\rfloor\right|\right|_\infty = \frac{1}{\Delta'}||e||_\infty \leq \frac{2^B}{2^{B+1}} = \frac{1}{2}.$$

  However, we additionally have the rounding error from the plaintext: the coefficients of $\frac{1}{\Delta'}\mathtt{pt}$ may not be integers. Thus, even though we can ensure that $||\frac{1}{\Delta'}e - \lceil \frac{1}{\Delta'}e \rfloor||_\infty \leq \frac{1}{2}$, due to the rounding error from the plaintext we may have $||(\frac{1}{\Delta'}\mathtt{pt} - \lceil \frac{1}{\Delta'}\mathtt{pt} \rfloor) + (\frac{1}{\Delta'}e - \lceil \frac{1}{\Delta'}e \rfloor)||_\infty \geq \frac{1}{2}$. In contrast to the noise, we cannot estimate how the rounding error for the plaintexts behaves, since the plaintexts can take any value. Thus, the rounding error of the plaintext may result in the rounding operation changing *all* plaintext bits of the message (even if the the absolute distance between the two may be small).
- Take $\mathsf{round}(\cdot) = \lfloor \cdot \rfloor$. We do not encounter the problem above as long as the coefficients of the error polynomial are positive. However, the error distribution is centered around zero, and thus we know that some coefficients will be negative. For these, we have $\mathtt{pt}_i + e_i \leq \mathtt{pt}_i$ and thus $\lfloor \mathtt{pt}_i + e_i \rfloor = \mathtt{pt}_i - 1$ for positive plaintext coefficients and $\lfloor \mathtt{pt}_i + e_i \rfloor = \mathtt{pt}_i + 1$ for negative ones. If the plaintext is close to a power of two, this can again change all the bits of the plaintext in its binary representation.

- Take $\mathsf{round}(\cdot) = \lceil \cdot \rceil$. Here we have a similar problem as above: decryption will be correct on all negative coefficients, but will fail on all the positive ones.

Therefore, we see that neither flooring nor ceiling can be used for our purposes. If we wish to use rounding to the nearest integer, then we need to guarantee that no additional rounding error comes from the plaintext. Otherwise put, we require that $\frac{1}{\Delta'}\mathsf{pt}$ is a polynomial with integer coefficients. This removes the problem outlined above. Indeed, we then have $||(\frac{1}{\Delta'}\mathsf{pt} - \lceil \frac{1}{\Delta'}\mathsf{pt} \rfloor + (\frac{1}{\Delta'}e - \lceil \frac{1}{\Delta'} \rfloor))||_\infty = ||0 + \frac{1}{\Delta'}e - \lceil \frac{1}{\Delta'}e \rfloor||_\infty$, for which we can easily find a Δ' such that this expression is smaller than $\frac{1}{2}$. The condition that $\frac{1}{\Delta'}\mathsf{pt}$ is a polynomial with integer coefficients is equivalent to the $\log_2(\Delta')$ lowest bits of the plaintext being 0. Since those are the error bits, this effectively means that we can remove those efficiently, as long as the message and error bits are separated within the ciphertext. We formalise this below.

5.1 Correctable Circuits

Definition 4 (Condition for Δ'-Correctability (cf. [CCH+22]))**.** Let $\mathsf{pt} \in \mathbb{Z}[i]^N$ be a plaintext, and let Δ' be some correction factor. We say that pt is Δ'-correctable if $\frac{1}{\Delta'}\mathsf{pt} \in \mathbb{Z}[i]^N$. We say that a circuit $f : (\mathbb{Z}[i]^N)^\ell \to \mathbb{Z}[i]^N$ is Δ'-correctable if $f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell)$ is Δ'-correctable for all choices of inputs pt_i.

If the conditions of the definition apply, then we can achieve (δ, r)-exact CKKS with our techniques. This leads to the following questions: how restrictive is the Δ'-correctable condition?

The CKKS scheme can evaluate circuits that can be represented as a polynomial. Therefore, we can write a circuit f as:

$$f(x_0, \ldots, x_\ell) = \sum_{i_0=0}^{k_0} \sum_{i_1=0}^{k_1} \cdots \sum_{i_\ell}^{k_\ell} a_{i_0 i_1 \ldots i_\ell} x_0^{i_0} x_1^{i_1} \ldots x_\ell^{i_\ell},$$

where $a_{i_0 i_1 \ldots i_\ell} \in \mathbb{Z}[i]^N$ in the plaintext space. Thus, f is Δ'-correctable if

$$\begin{aligned} &\frac{1}{\Delta'} f(\mathsf{pt}_0, \ldots, \mathsf{pt}_\ell) \in \mathbb{Z}[i] \quad \forall \mathsf{pt}_i \in \mathbb{Z}[i] \\ \Leftrightarrow &\frac{1}{\Delta'} \sum_{i_1=0}^{k_1} \cdots \sum_{i_\ell}^{k_\ell} a_{i_0 i_1 \ldots i_\ell} \mathsf{pt}_0^{i_0} \mathsf{pt}_1^{i_1} \ldots \mathsf{pt}_\ell^{i_\ell} \in \mathbb{Z}[i] \\ \Leftrightarrow &\sum_{i_1=0}^{k_1} \cdots \sum_{i_\ell}^{k_\ell} \frac{1}{\Delta'} a_{i_0 i_1 \ldots i_\ell} \mathsf{pt}_0^{i_0} \mathsf{pt}_1^{i_1} \ldots \mathsf{pt}_\ell^{i_\ell} \in \mathbb{Z}[i] \end{aligned}$$

This can be achieved by scaling up the coefficients of the circuit by Δ', and scaling down by Δ' before outputting the result of the computation. Note that scaling the plaintext by Δ' does not guarantee the correct result, since there might be constant terms that are not multiples of Δ'. We investigate this.

5.2 $(\delta, 0)$-exact CKKS

We introduce the following notation. Let $f : ((\mathbb{Z}[i])^N)^\ell \to (\mathbb{Z}[i])^N$ be a polynomial in ℓ variables over the plaintext space. Let C_f be the corresponding circuit on the ciphertext space. Let $f_{\Delta'} = \Delta' f$ and $C_{f_{\Delta'}}$ be the corresponding circuit on the ciphertext space. We define $\mathsf{CKKS}_{\delta,0}$. All operations that are not explicitly described are the same as in CKKS. To simplify notation, we omit the zero in the index. We denote by $\mathcal{P}_\delta$ the plaintext space and by $\mathcal{C}_\delta$ the admissible circuit space of CKKS_δ respectively.

Plaintext Space We have $\mathcal{P}_\delta = \mathbb{Z}[i]^N$.

Admissible Circuits The admissible circuit space of CKKS_δ is

$$\mathcal{C} = \{C_f | f : ((\mathbb{Z}[i])^N)^\ell \to (\mathbb{Z}[i])^N \text{can be represented as a polynomial }\}.$$

Evaluation Compute $f_{\Delta'} = \Delta' f$ and the corresponding $C_{f_{\Delta'}}$. Then return $\mathsf{Eval}_\delta(C_f, \mathtt{ct}_0, \ldots, \mathtt{ct}_\ell, \Delta', \mathtt{evk}) = \mathsf{Eval}(C_{f_{\Delta'}}, \mathtt{ct}_0, \ldots, \mathtt{ct}_\ell, \Delta', \mathtt{evk})$.

Decryption We have $\mathsf{Dec}_\delta(\mathtt{ct}, \mathtt{sk}) = \left\lceil \frac{1}{\Delta'} \mathsf{Dec}_{\mathsf{CKKS}}(\mathtt{ct}, \mathtt{sk}) \right\rfloor$.

We proceed by proving that the scheme is $(\delta, 0)$-exact.

Lemma 17. *Let CKKS_δ be the variant of the CKKS scheme as defined above. CKKS_δ is $(\delta, 0)$-exact, if the noise term e is bounded by $||e||_\infty \leq \frac{\Delta'}{2}$ with probability δ.*

Proof. Let $\mathtt{ct} = \mathsf{Eval}_\delta(C_f, \mathtt{ct}_0, \ldots, \mathtt{ct}_\ell, \Delta', \mathtt{evk})$ be a ciphertext, let $\mathtt{ct}_0, \ldots, \mathtt{ct}_\ell$ be fresh ciphertexts encrypting plaintexts $\mathtt{pt}_0, \ldots, \mathtt{pt}_\ell$ and let the circuit $C_f \in \mathcal{C}$. Note that C_f can be the identity circuit and therefore $\mathtt{ct}$ could also be a fresh ciphertext. Then we have

$$\begin{aligned}
\mathtt{Decode}_\delta(\mathtt{ct}, \Delta', \mathtt{sk}) &= \left\lceil \frac{1}{\Delta'} \mathtt{Decode}(\mathtt{ct}, \mathtt{sk}) \right\rfloor = \left\lceil \frac{1}{\Delta'} \mathtt{Decode}(C_{f_{\Delta'}}(\mathtt{ct}_0, \ldots, \mathtt{ct}_\ell), \mathtt{sk}) \right\rfloor \\
&= \left\lceil \frac{1}{\Delta'} (f_{\Delta'}(\mathtt{pt}_0, \ldots, \mathtt{pt}_\ell) + e) \right\rfloor = \left\lceil \frac{1}{\Delta'} (\Delta' f(\mathtt{pt}_0, \ldots, \mathtt{pt}_\ell) + e) \right\rfloor \\
&= \left\lceil f(\mathtt{pt}_0, \ldots, \mathtt{pt}_\ell) + \frac{1}{\Delta'} e \right\rfloor = f(\mathtt{pt}_0, \ldots, pt_\ell) + \left\lceil \frac{1}{\Delta'} e \right\rfloor .
\end{aligned}$$

That is

$$\mathtt{Decode}_\delta(\mathtt{ct}, \Delta', \mathtt{sk}) = f(\mathtt{pt}_0, \ldots, \mathtt{pt}_\ell) \Leftrightarrow \left\lceil \frac{1}{\Delta'} e \right\rfloor = 0 \Leftrightarrow \left|\left| \frac{1}{\Delta'} e \right|\right|_\infty \leq \frac{1}{2} \Leftrightarrow ||e||_\infty \leq \frac{\Delta'}{2}.$$

□

The last step is to derive the probability that $||e||_\infty \leq \frac{\Delta'}{2}$ holds. For plain CKKS, the error growth can be tightly bounded by our results presented in Section 3. In this $(\delta, 0)$-exact version of CKKS, the constant terms are scaled by an additional factor Δ'. This means that the noise resulting from a multiplication by a constant operation will also be larger by a factor of Δ'. If no noise management operations are used during the computation of a circuit C_f, scaling by a factor Δ' results in the noise also being a factor Δ' larger than the regular CKKS noise. Thus, the condition $\frac{1}{\Delta'}||e||_\infty \leq \frac{1}{2}$ is very unlikely to hold. For the condition to hold, we would need to increase Δ', resulting in a larger noise growth and so on. However, we showed in Section 3 that after rescaling, the overall noise is dominated by a rounding error. Thus, by applying rescaling in strategic points in the circuit and using optimisations (such as multiplying by the binary decomposition of Δ'), the noise growth in the exact version can reduced to the one in regular CKKS. Thus, the condition can be achieved.

5.3 Experimental Results

We showed in Section 3 that the noise bounds hold with probability $\delta \approx \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$ for n the number of plaintext slots, and α a parameter (usually $\alpha \in [6, 14]$). The parameter α determines the tightness of the bounds – the smaller α is, the tighter the bounds are, but the larger the failure probability is. α therefore introduces a trade-off between tightness and correctness. By applying our bounds to determine the number of bits to round off,

we can obtain $(\delta, 0)$-exact CKKS. Our $(\delta, 0)$-exact CKKS scheme retains the efficiency advantages over schemes such as BGV and BFV. However, the precision that we can obtain is reduced. Indeed, since we require that the lowest $r = \lceil \log_2(B) \rceil$ bits be zero for some error bound B, this means that to obtain the same precision as regular CKKS, we need to choose the precision factor Δ to be B bits larger. In the following, we present experimental results on our $(\delta, 0)$-exact CKKS scheme.

$\log(\Delta')$	0	1	2	3	4	5	6	7	8	9	10	11	12
δ	0.00105	0.00314	0.00754	0.0161	0.0330	0.0655	0.130	0.254	0.462	0.698	0.850	0.926	0.964
$\log(\Delta')$	13	14	15	16	17	18	19	20	21	22	23	24	25
δ	0.981	0.991	0.995	0.998	0.99903	0.99948	0.99972	0.99990	0.99991	0.99997	0.99999	1	1

Figure 1: Concrete instantiation of $(\delta, 0)$-CKKS using the setup of experiment II (High Degree Chebyshev Approximation of the Sigmoid, Section 4.3) with $\Delta = 55$ and $h = 2N/3$. We set $\mathsf{round}(\cdot) = \lceil \cdot \rfloor$. We quantize all input and output values in the ring by Δ'. That is, plaintexts before being encrypted, plaintexts after being decrypted, and the coefficients of the Chebyshev polynomial. We see that we achieve $\delta = 1$ when $\log(\Delta') = 24$.

5.4 Exact CKKS and IND-CPA-D Security

We can easily see that a $(\delta, 0)$-exact CKKS scheme is IND-CPA-D secure if $1-\delta$ is negligible. There are common parameter sets such that $1 - \delta$ is negligible. Choosing for example $n = 8192$ and $\alpha = 14$ results in a negligible δ, as does choosing $\alpha = 14$ and $n = 16384$. However, the above results do not directly give us IND-CPA-D security. The reason is that, in order to bound the error in CKKS, we must approximate the distribution of the error by a normal distribution – this is well founded, the central limit theorem indeed gives us that the distribution converges towards a normal distribution, and from experimental evaluation we know that this convergence is fast. However, the distribution is only *approximately* Gaussian, and the actual distance from a normal distribution is hard to calculate. Thus, we cannot say by how much the distribution fails to be normal. We know that if the distribution is normal, the bounds hold with probability δ. But in order to calculate the advantage an adversary has against the IND-CPA-D game, we also need to compute the probability that the distribution fails to be normal, which would give us the (full) probability that the bounds fail. This probability strongly depends on the circuit that is evaluated, leading to circuit-specific intractable calculations. Experimentally, we know that the probability that the distribution fails to behave in its tails like a normal distribution is small. Thus, applying our procedure is likely to prevent IND-CPA-D attacks in practice. However, formally this cannot be proven, since the complete failure probability is hard to quantify. This problem can be circumvented by using worst-case bounds that hold with probability 1: instead of making any assumptions about the distribution, the error term can also be bounded by looking the absolute maximal value of each element in the error term. However, these bounds are so large that applying them would also mean losing the entire message.

6 Conclusion

In this work, we present tight average-case noise bounds for all basic operations in CKKS. We show that our bounds are composable, thus allowing us to provide tight noise estimates for complex homomorphic circuits. Additionally, we provide a tool – the estimator – that allows us to illustrate how to combine our estimates for basic operations into estimates for arbitrarily large circuits. Further, we provide experimental results showing the tightness of our bounds. Given a circuit (such as binary value cleaning and bootstrapping), we run

it homomorphically, and we also run the estimator on it. Then, we compare the distance between noise output of the former and the estimate output of the latter. We find that our estimates deviate by less than 0.01 bits from the experimentally observed noise. Thus, we provide the first tight noise analysis for CKKS bootstrapping, and more generally, the first tight noise analysis for arbitrary circuits in CKKS. Additionally, we discuss an application of our work: we show how our results can be used to obtain an exact version of the CKKS scheme that preserves the efficiency advantages CKKS has over schemes such as BGV and BFV. We additionally show how such an exact CKKS scheme is a step towards a practical solution for achieving IND-CPA-D security for CKKS.

Acknowledgments

Christian Mouchet was supported by the German Federal Ministry of Research, Technology and Space (BMFTR) through the funding of the ATLAS project (ref.# 16KISA037). Tune Insight's work in this article is partially funded by the European Union under Grant Agreement No. 101167964. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the European Cybersecurity Competence Centre can be held responsible for them.

References

[AKP24] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. Cryptology ePrint Archive, Paper 2024/1623, 2024. URL: https://eprint.iacr.org/2024/1623.

[BCDS+25] Flavio Bergamaschi, Anamaria Costache, Dana Dachman-Soled, Hunter Kippen, Lucas LaBuff, and Rui Tang. Revisiting the security of approximate FHE with noise-flooding countermeasures. In *IACR International Conference on Public-Key Cryptography*, pages 102–134. Springer, 2025. doi:10.1007/978-3-031-91832-2_4.

[BCKS24] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In *Eurocrypt*, pages 94–123. Springer, 2024. doi:10.1007/978-3-031-58723-8_4.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090262.

[BKSS25] Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 330–360. Springer, 2025. doi:10.1007/978-981-96-0875-1_11.

[BMCM23] Beatrice Biasioli, Chiara Marcolla, Marco Calderini, and Johannes Mono. Improving and automating BFV parameters selection: An average-case approach. Cryptology ePrint Archive, Paper 2023/600, 2023. URL: https://eprint.iacr.org/2023/600.

[BMTPH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic

encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-77870-5_21.

[BTPH22] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 521–541, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-09234-3_26.

[CCH+22] Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2022/162, 2022. Version 20220220:200321 of the paper. URL: https://eprint.iacr.org/2022/162.

[CCH+23] Anamaria Costache, Benjamin R Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 325–345. Springer, 2023. doi:10.1007/978-3-031-53368-6_16.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17656-3_2.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1), 2020. doi:10.1007/s00145-019-09319-x.

[CHH18] Jung Hee Cheon, Kyoohyung Han, and Minki Hhan. Faster homomorphic discrete Fourier transforms and improved FHE bootstrapping. Cryptology ePrint Archive, Paper 2018/1073, 2018. https://eprint.iacr.org/2018/1073. URL: https://eprint.iacr.org/2018/1073.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-78381-9_14.

[CHK+19] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-10970-7_16.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-70694-8_15.

[CLP20] Anamaria Costache, Kim Laine, and Rachel Player. Evaluating the effectiveness of heuristic worst-case noise analysis in FHE. In Liqun Chen, Ninghui

Li, Kaitai Liang, and Steve Schneider, editors, *Computer Security – ESORICS 2020*, pages 546–565, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-59013-0_27.

[CNP23] Anamaria Costache, Lea Nürnberger, and Rachel Player. Optimisations and tradeoffs for HElib. In Mike Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, pages 29–53, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-30872-7_2.

[CS16] Ana Costache and Nigel P Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Topics in Cryptology-CT-RSA 2016: The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29-March 4, 2016, Proceedings*, pages 325–340. Springer, 2016. doi:10.1007/978-3-319-29485-8_19.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-46800-5_24.

[DMPS24] N. Drucker, G. Moshkowich, T. Pelleg, and H. Shaul. BLEACH: Cleaning errors in discrete computations. *Journal of Cryptology*, 2024. doi:10.1007/s00145-023-09483-1.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144. URL: https://eprint.iacr.org/2012/144.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*. Springer, 2013. doi:10.1007/978-3-642-40041-4_5.

[GZ25] Mingyu Gao and Hongren Zheng. A critique on average-case noise analysis in RLWE-based homomorphic encryption. Cryptology ePrint Archive, Paper 2025/1036, 2025. URL: https://eprint.iacr.org/2025/1036.

[HK20] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020*, 2020. doi:10.1007/978-3-030-40186-3_16.

[Ili19] Ilia Iliashenko. *Optimisations of Fully Homomorphic Encryption*. PhD thesis, KU Leuven, 2019.

[JM20] Charanjit S. Jutla and Nathan Manohar. Modular lagrange interpolation of the mod function for bootstrapping of approximate HE. Cryptology ePrint Archive, Paper 2020/1355, 2020. URL: https://eprint.iacr.org/2020/1355.

[JM22] Charanjit S Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate HE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 491–520. Springer, 2022. doi:10.1007/978-3-031-06944-4_17.

[Kim25] Jaehyung Kim. Efficient homomorphic integer computer from CKKS. Cryptology ePrint Archive, Paper 2025/066, 2025. URL: https://eprint.iacr.org/2025/066.

[KPK+22] Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. EvalRound algorithm in CKKS bootstrapping. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 161–187. Springer, 2022. doi:10.1007/978-3-031-22966-4_6.

[Lat24] Lattigo. Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

[LLK+22] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-06944-4_19.

[LLKN20] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Near-optimal polynomial for modulus reduction using L2-norm for approximate homomorphic encryption. *IEEE Access*, 8:144321–144330, 2020. doi:10.1109/ACCESS.2020.3014369.

[LLL+21] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-77870-5_22.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-77870-5_23.

[MP24] Sean Murphy and Rachel Player. A central limit approach for Ring-LWE noise analysis. *IACR Communications in Cryptology*, 1(2), 2024. doi:10.62056/ay76c0kr.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009. doi:10.1145/1568318.1568324.

Auxiliary Supplemental Material

A Proofs of the Lemmas in Sections 2 and 3

Lemma 2 (Squaring of Random Polynomials). *Let $f \in \mathcal{R}$ be a polynomial of even degree N, whose coefficients are drawn identically and independently from a symmetric distribution $\mathcal{D}$ with standard deviation σ_f and expectation 0. Then we have for the standard deviation of $(f^2)_i, i \in \{0, \ldots, N-1\}$*

$$\begin{cases} \sigma_{(f^2)_i} = \sigma_f^2\sqrt{2N+1} & \text{, for } i \text{ even} \\ \sigma_{(f^2)_i} = \sigma_f^2\sqrt{2N} & \text{, for } i \text{ odd.} \end{cases}$$

Proof. Since $f \in \mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, following a result given by [Ili19], we can write the coefficients of f^2 as follows

$$(f^2)_i = \sum_{k=0}^{i} f_k f_{i-k} - \sum_{k=i+1}^{N-1} f_k f_{N-k+i}.$$

Now, for $i = 0$ and n even we get

$$(f^2)_0 = (f_0)^2 - f_1 f_{N-1} - f_2 f_{N-2} - \ldots - f_{N-2} f_2 - f_{N-1} f_1.$$

We can write the above as

$$\begin{aligned} (f^2)_0 &= (f_0)^2 - (f_1 f_{N-1} + \ldots + f_{N-1} f_1) \\ &= (f_0)^2 - \sum_{k=1}^{\frac{N}{2}-1} 2 f_k f_{N-k} + (f_{\frac{N}{2}})^2. \end{aligned}$$

For $i \neq 0$ and i even, we get

$$\begin{aligned} (f^2)_i =& f_0 f_i + f_1 f_{i-1} + \ldots + f_{i-1} f_1 + f_i f_0 - f_{i+1} f_{N-1} - f_{i+2} f_{N-2} \\ & - \ldots - f_N f_{i+1} \\ =& \sum_{k=0}^{\frac{i}{2}} 2 f_k f_{i-k} + \sum_{k=i+1}^{\frac{N}{2}} 2 f_k f_{N-k+i}. \end{aligned}$$

Similarly, for i odd, we get

$$(f^2)_i = \sum_{k=0}^{\lfloor \frac{i}{2} \rfloor} 2 f_k f_{i-k} - \sum_{k=i+1}^{\frac{N}{2}} 2 f_k f_{N+i-k}.$$

In the above form, all the terms are again independent. This is since the coefficients are independently drawn, and there are no cross terms. We can therefore calculate the coefficient variance $(f^2)_i$ from the coefficient variance σ_f^2 of f as follows.

Each of the terms $f_i f_j$ for $i, j \in \{0, \ldots, N-1\}$ has variance σ_f^4. For $i = 0$, we have $\frac{N}{2}$ terms of the form $2 f_i f_j$ with variance $4\sigma_f^2$, and one term which is not a product of two terms and therefore has variance σ_f^2. Therefore, for the variance of $(f^2)_0$ we get

$$\sigma^2_{(f^2)_0} = \frac{4N}{2}\sigma_f^4 + \sigma_f^4 = (2N+1)\sigma_f^4.$$

When i is even and $i \neq 0$, each of the remaining terms appeared twice and we end up with $\frac{N}{2}$ independent terms, each of which has variance $4\sigma_f^4$ and the term $(f_{\frac{N}{2}})^2$, which has variance σ_f^4. Therefore, we get

$$\sigma^2_{(f^2)_i} = (2N+1)\sigma_f^4.$$

Lastly, for i odd we have $\frac{N}{2}$ with variance $4\sigma_f^2$ and therefore get

$$\sigma^2_{(f^2)_i} = 2N\sigma_f^4.$$

Thus we get for the standard deviation for f_i for i even $\sigma_{(f^2)_i} = \sqrt{2N+1}\sigma_f^2$ and for i odd we get $\sigma_{(f^2)_i} = \sqrt{2N}\sigma_f^2$. □

Lemma 6 (Encoding and Fresh Encryption Noise). *Let* $\mathbf{m}$ *be a message with scaling factor* Δ*, and let* $\mathtt{pt} \in \mathcal{R}_{Q_\ell}$ *be the encoding of* $\mathbf{m}$*. Let* $\mathtt{ct}_{pk}$ *be a public-key RNS-CKKS ciphertext, and let* $\mathtt{ct}_{sk}$ *be a secret key RNS-CKKS ciphertext, both encrypting* $\mathtt{pt}$ *using the respective* `Encrypt()` *function. Let* σ_0 *be the standard deviation of the error distribution. Then we have for the coefficient standard deviation of component-wise noise*

$$\sigma_{n(\mathtt{pt})} = \sqrt{\frac{1}{12}} \qquad \sigma_{n(\mathtt{ct}_{\mathtt{pk}})} = \left(\sqrt{\frac{1}{6}}, \sqrt{\frac{1}{12}}\right) \qquad \sigma_{n(\mathtt{ct}_{\mathtt{sk}})} = \left(\sqrt{\sigma_0^2}, 0\right).$$

Proof. The noise in the plaintext stems from the rounding operation after scaling the plaintext. The rounding operation rounds each coefficient to the nearest integer. The noise in each component is therefore the difference between the raw value and the rounded value. The difference is a polynomial $\tau^{(j)}$, with coefficients that are continuously uniformly distributed in $\left[\frac{1}{2}, \frac{1}{2}\right)$.

We have for the structure of a freshly encrypted ciphertext

$$\begin{aligned}
\mathtt{ct} &= \left(\left\lceil P^{-1}[\mathtt{pk}[0]u + e^{(0)}]_{QP}\right\rfloor + \mathtt{pt} + n(\mathtt{pt}), \left\lceil P^{-1}[\mathtt{pk}[1]u + e^{(1)}]_{QP}\right\rfloor\right) \\
&= \left(\left\lceil P^{-1}[-asu + e_{\mathtt{pk}}u + e^{(0)}]_{QP}\right\rfloor + \mathtt{pt} + n(\mathtt{pt}), \left\lceil P^{-1}[au + e^{(1)}]_{QP}\right\rfloor\right) \\
&= ([-bs + \tau^{(0)} + \mathtt{pt} + n(\mathtt{pt})]_Q, [b + \tau^{(1)}]_Q).
\end{aligned}$$

All the terms that are multiplied by the mask $b = P^{-1}au$ will be cancelled out during decryption, therefore the component-wise noise consists of

$$n(\mathtt{ct}) = (\tau^{(0)} + n(\mathtt{pt}), \tau^{(1)}),$$

as long as $||(e_{\mathtt{pk}}u + e^{(0)}, e^{(1)})||_\infty \leq P/2$, which is the case with high probability because $e_{\mathtt{pk}}, e^{(0)}, e^{(1)}$ are drawn from the error distribution and therefore have standard deviations σ_0 and u is drawn from the secret key distribution with Hamming weight h and therefore has standard deviation $\sqrt{\frac{h}{N}}$.

By Lemma 1 and the additivity of variances we therefore have

$$\sigma_{n(\mathtt{ct})} = \left(\sigma_{(n(\mathtt{ct})[0])}, \sigma_{(n(\mathtt{ct})[1])}\right) = \left(\sqrt{\tfrac{1}{12} + \sigma^2_{n(\mathtt{pt})}}, \sqrt{\tfrac{1}{12}}\right) = \left(\sqrt{\tfrac{1}{6}}, \sqrt{\tfrac{1}{12}}\right).$$

The secret key ciphertext is created as $\mathtt{ct}_{\mathtt{sk}} = (-as + e, a)$, where $a \xleftarrow{\$} \mathcal{R}_Q$, and $e \leftarrow \chi$ is drawn from the error distribution with standard deviation σ_0. Since the terms multiplied by a cancel out, e is the only error term left. The result follows. □

Lemma 12 (Tensor Product). *Assume* $\mathtt{ct}^{(0)} \neq \mathtt{ct}^{(1)}$. *Let* $\{\mathtt{ct}^{(2)}, \min(Q_{\ell_0}, Q_{\ell_1}), \Delta_0\Delta_1\} = \mathit{Tensor}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{ct}^{(1)}, Q_{\ell_1}, \Delta_1\})$, *and let* $\{\mathtt{ct}^{(3)}, Q_{\ell_0}, \Delta_0^2\} = \mathit{Tensor}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\})$, *the result of a squaring. Then we have for the coefficient standard deviation of the component-wise noise*

$$\begin{aligned}
&\sigma_{n(\mathtt{ct}^{(2)})} \\
&= \Bigg(\sqrt{N\left(\sigma^2_{\mathtt{pt}^{(0)}}\sigma^2_{n(\mathtt{ct}^{(1)})[0]} + \sigma^2_{\mathtt{pt}^{(1)}}\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{n(\mathtt{ct}^{(1)})[0]}\right)}, \\
&\quad \sqrt{N\left(\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left(\sigma^2_{\mathtt{pt}^{(1)}} + \sigma^2_{n(\mathtt{ct}^{(1)})[1]}\right) + \sigma^2_{n(\mathtt{ct}^{(1)})[1]}\left(\sigma^2_{\mathtt{pt}^{(0)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[1]}\right)\right)}, \\
&\quad \sqrt{N}\sigma_{n(\mathtt{ct}^{(0)})[1]}\sigma_{n(\mathtt{ct}^{(1)})[1]} \Bigg).
\end{aligned}$$

$$\begin{aligned}
&\sigma^2_{n(\mathtt{ct}^{(3)})} \\
&= \Bigg(\sqrt{2N\left(2\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{\mathtt{pt}^{(0)}} + \sigma^4_{n(\mathtt{ct}^{(0)})[0]}\right)}, \sqrt{4N\left(\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\left(\sigma^2_{\mathtt{pt}^{(0)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[0]}\right)\right)}, \\
&\quad \sqrt{2N}\sigma^2_{n(\mathtt{ct}^{(0)})[1]} \Bigg).
\end{aligned}$$

Proof. $\mathtt{ct}^{(2)}$ is calculated as $(\mathtt{ct}^{(0)}[0]\mathtt{ct}^{(1)}[0], \mathtt{ct}^{(0)}[0]\mathtt{ct}^{(1)}[1] + \mathtt{ct}^{(0)}[1]\mathtt{ct}^{(1)}[0], \mathtt{ct}^{(0)}[1]\mathtt{ct}^{(1)}[1])$. We denote $\mathtt{ct}^{(i)}$ as $(\mathtt{pt}^{(i)} + a^{(i0)} + n(\mathtt{ct}^{(i)})[0], a^{(i1)} + n(\mathtt{ct}_i)[1])$, where $a^{(ij)}$ is some mask that gets cancelled out during decryption. Since $\mathtt{ct}^{(0)}, \mathtt{ct}^{(1)}$ are not necessarily fresh ciphertexts, it no longer makes sense to consider the encoding noise $n(\mathtt{pt}^{(i)})$ separately. Instead, this is a part of the component-wise ciphertext noise $n(\mathtt{ct}^{(i)}[j])$. Thus, we have

$$\begin{aligned}
&\mathtt{ct}^{(0)}[0]\mathtt{ct}^{(1)}[0] \\
&= (\mathtt{pt}^{(0)} + a^{(00)} + n(\mathtt{ct}^{(0)})[0])(\mathtt{pt}^{(1)} + a^{(10)} + n(\mathtt{ct}^{(1)})[0]) \\
&= \mathtt{pt}^{(0)}\mathtt{pt}^{(1)} + \mathtt{pt}^{(0)}a^{(10)} + \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[0] + a^{(00)}\mathtt{pt}^{(1)} \\
&\quad + a^{(00)}a^{(10)} + a^{(00)}n(\mathtt{ct}^{(1)})[0] + n(\mathtt{ct}^{(0)})[0]\mathtt{pt}^{(1)} \\
&\quad + n(\mathtt{ct}^{(0)})[0]a^{(10)} + n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[0] \\
&= \mathtt{pt}_\times + a^{(0)\times} + \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[0] + \mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[0] + n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[0],
\end{aligned}$$

where $a^{(0)\times} = \mathtt{pt}^{(0)}a^{(10)} + a^{(00)}\mathtt{pt}^{(1)} + a^{(00)}a^{(10)} + a^{(00)}n(\mathtt{ct}^{(1)})[0] + a^{(10)}n(\mathtt{ct}^{(0)})$ is the collection of all the terms containing the mask, which will therefore be cancelled out. Therefore, we get for the component-wise noise of $n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[0]$

$$n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[0] = n(\mathtt{pt}_\times) + \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[0] + \mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[0] + n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[0].$$

Calculating the second component of the resulting ciphertext similarly gives

$$\begin{aligned}
&\mathtt{ct}^{(0)}[0]\mathtt{ct}^{(1)}[1] + \mathtt{ct}^{(0)}[1]\mathtt{ct}^{(1)}[0] \\
&= (\mathtt{pt}^{(0)} + a^{(00)} + n(ct_0)[0])(a^{(11)} + n(\mathtt{ct}^{(1)})[1]) \\
&\quad + (a^{(01)} + n(\mathtt{ct}^{(0)})[1])(\mathtt{pt}^{(1)} + a^{(10)} + n(\mathtt{ct}^{(1)})[0])
\end{aligned}$$

$$
\begin{aligned}
&= \mathtt{pt}^{(0)}a^{(11)} + a^{(00)}a^{(11)} + n(\mathtt{ct}^{(0)})[0]a^{(11)} + \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[1] + a^{(00)}n(\mathtt{ct}^{(1)})[1] \\
&\quad + n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[1] + a^{(01)}\mathtt{pt}^{(1)} + n(\mathtt{ct}^{(0)})[1]\mathtt{pt}^{(1)} + a^{(01)}a^{(10)} \\
&\quad + n(\mathtt{ct}^{(0)})[1]a^{(10)} + a^{(01)}n(\mathtt{ct}^{(1)})[0] + n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[1] \\
&= a^{(1)\times} + \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[1] + \mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[1] + n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[1] \\
&\quad + n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[0],
\end{aligned}
$$

where $a^{(1)\times}$ again is the sum of all the terms containing the masks, and therefore is cancelled out. Then we get for the component-wise noise of the second component of the multiplication result

$$
\begin{aligned}
n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[1] = \mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[1] + \mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[1] + n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[0] \\
+ n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[1].
\end{aligned}
$$

For the last element in the ciphertext after the tensor product we get

$$
\begin{aligned}
\mathtt{ct}^{(0)}[1]\mathtt{ct}^{(1)}[1] &= (a^{(01)} + n(\mathtt{ct}^{(0)})[1])(a^{(11)} + n(\mathtt{ct}^{(1)})[1]) \\
&= a^{(01)}a^{(11)} + a^{(10)}n(\mathtt{ct}^{(1)})[1] + n(\mathtt{ct}^{(0)})[1]a^{(11)} + n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[1] \\
&\Rightarrow n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[2] = n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[1].
\end{aligned}
$$

Since the mask a is cancelled out during decryption, the last line follows.

Thus we get for the coefficients variances

$$
\begin{aligned}
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[0]} &= \sigma^2_{(\mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[0])} + \sigma^2_{(\mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[0])} + \sigma^2_{(n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[0])} \\
&= N\sigma^2_{n(\mathtt{ct}^{(1)})[0]}\sigma^2_{\mathtt{pt}^{(0)}} + N\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{\mathtt{pt}^{(1)}} + N\sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{n(\mathtt{ct}^{(1)})[0]},
\end{aligned}
$$

$$
\begin{aligned}
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[1]} &= \sigma^2_{(\mathtt{pt}^{(0)}n(\mathtt{ct}^{(1)})[1])} + \sigma^2_{(\mathtt{pt}^{(1)}n(\mathtt{ct}^{(0)})[1])} + \sigma^2_{(n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(1)})[1])} \\
&\quad + \sigma^2_{(n(\mathtt{ct}^{(0)})[1]n(\mathtt{ct}^{(1)})[0])} \\
&= N\big(\sigma^2_{n(\mathtt{ct}^{(1)})[1]}\sigma^2_{\mathtt{pt}^{(0)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[1]}\sigma^2_{\mathtt{pt}^{(1)}} + \sigma^2_{n(\mathtt{ct}^{(0)})[0]}\sigma^2_{n(\mathtt{ct}^{(1)})[1]} \\
&\quad + \sigma^2_{n(\mathtt{ct}^{(0)})[1]}\sigma^2_{n(\mathtt{ct}^{(1)})[0]}\big),
\end{aligned}
$$

$$
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(1)})[2]} = N\sigma^2_{n(\mathtt{ct}^{(0)})[1]}\sigma^2_{n(\mathtt{ct}^{(1)})[1]}.
$$

Lastly, for squaring we can calculate the component-wise noise similarly. We get

$$
\begin{aligned}
&\mathtt{ct}^{(0)}[0]\mathtt{ct}^{(0)}[0] \\
&\quad = (\mathtt{pt}^{(0)})^2 + (a^{(00)})^2 + (n(\mathtt{ct}^{(0)})[0])^2 + 2a^{(00)}(\mathtt{pt}^{(0)} + n(\mathtt{ct}^{(0)})[0]\mathtt{pt}^{(0)} \\
&\quad + a^{(00)}n(\mathtt{ct}^{(0)})[0]) \\
&\Rightarrow n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})[0] = n(\mathtt{ct}^{(0)})[0]^2 + 2n(\mathtt{ct}^{(0)})[0]\mathtt{pt}^{(0)},
\end{aligned}
$$

where the last line again follows from the fact that all terms that are a multiple of a cancelling out during decryption.

$$
2\mathtt{ct}^{(0)}[0]\mathtt{ct}^{(0)}[1] = 2(\mathtt{pt}^{(0)} + a^{(00)} + n(\mathtt{ct}^{(0)})[0])(a^{(01)} + n(\mathtt{ct}^{(0)})[1])
$$

$$
\begin{aligned}
&= 2\mathtt{pt}^{(0)}a^{(10)} + 2\mathtt{pt}^{(0)}n(\mathtt{ct}^{(0)})[1] + 2a^{(00)}a^{(01)} \\
&\quad + 2a^{(00)}n(\mathtt{ct}^{(0)})[1] + 2n(\mathtt{ct}^{(0)})[0]a^{(01)} + 2n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(0)})[1],
\end{aligned}
$$

$$
\Rightarrow n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})[1] = 2\mathtt{pt}^{(0)}n(\mathtt{ct}^{(0)})[1] + 2n(\mathtt{ct}^{(0)})[0]n(\mathtt{ct}^{(0)})[1],
$$

$$
\begin{aligned}
\mathtt{ct}^{(0)}[1]\mathtt{ct}^{(0)}[1] &= (a^{(01)} + n(\mathtt{ct}^{(0)})[1])^2 \\
&= (a^{(01)})^2 + 2a^{(01)}n(\mathtt{ct}^{(0)})[1] + n(\mathtt{ct}^{(0)})[1]^2,
\end{aligned}
$$

$$
\Rightarrow n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})[2] = n(\mathtt{ct}^{(0)})[1]^2.
$$

Therefore, we get for the coefficient variances

$$
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})_i[0]} \overset{Lemma\ 2}{=} 2N\sigma^4_{n(\mathtt{ct}^{(0)})_i[0]} + 4N\sigma^2_{n(\mathtt{ct}^{(0)})_i[0]}\sigma^2_{\mathtt{pt}_i^{(0)}} + 4N\sigma^2_{\mathtt{pt}_i^{(0)}}
$$

$$
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})_i[1]} = 4N\sigma^2_{n(\mathtt{ct}^{(0)})_i[1]}\sigma^2_{\mathtt{pt}_i^{(0)}} + 4N\sigma^2_{n(\mathtt{ct}^{(0)})_i[0]}\sigma^2_{n(\mathtt{ct}^{(0)})_i[1]}
$$

$$
\sigma^2_{n(\mathtt{ct}^{(0)}\mathtt{ct}^{(0)})_i[2]} = 2N\sigma^4_{n(\mathtt{ct}^{(0)})_i[1]}.
$$

□

Lemma 13 (Key-Switch). *Let* $\mathtt{ct}' = (\mathtt{ct}'[0], \mathtt{ct}'[1])$ *be a ciphertext with respect to a secret key* $(1, s')$ *and let* $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) = \mathtt{KeySwitch}(\mathtt{ct}', s', s)$ *be the key switched ciphertext now with respect to a secret key* $(1, s)$*. Then for the component-wise noise of* $\mathtt{ct}$ *we have*

$$
n(\mathtt{ct}) = n(\mathtt{ct}') + \left(P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}} n(\mathtt{evk})[0] + \tau_0, \tau_1 \right),
$$

and for the coefficient standard deviation of $n(\mathtt{ct})$

$$
\sigma_{n(\mathtt{ct})}) = \left(\sqrt{\sigma^2_{n(\mathtt{ct}')[0]} + \frac{\sum_{i=0}^{\beta} q_{\gamma_i}^2\sigma_0^2}{12P^2} + \frac{1}{12}}, \sqrt{\sigma^2_{n(\mathtt{ct}')[1]} + \frac{1}{12}} \right).
$$

Proof. We calculate $\mathtt{ct}$ as follows: decompose $\mathtt{ct}'[1]$ into digits $[\mathtt{ct}'[1]]_{q_{\gamma_i}}$ with respect to a basis $\left(\frac{Q_L}{q_{\gamma_i}}\right)_{0\le i\le\beta}$, such that $\left\langle [\mathtt{ct}'[1]]_{q_{\gamma_{0\le i\le\beta}}}, \left(\frac{Q_L}{q_{\gamma_i}}\left[\left(\frac{Q_L}{q_{\gamma_i}}\right)^{-1}\right]_{q_{\gamma_i}}\right)_{0\le i\le\beta}\right\rangle = \mathtt{ct}'[1]$. Then we calculate

$$
\begin{aligned}
\mathtt{ct}[0] &= \mathtt{ct}'[0] + \left\lceil P^{-1}\langle [\mathtt{ct}'[1]]_{q_{\gamma_{0\le i\le\beta}}}, \mathtt{evk}[0]\rangle \right\rfloor \\
\mathtt{ct}[1] &= \left\lceil P^{-1}\langle [\mathtt{ct}'[1]]_{q_{\gamma_{0\le i\le\beta}}}, \mathtt{evk}[1]\rangle \right\rfloor.
\end{aligned}
$$

To see which terms are cancelled out, it is easier to first calculate the term $\mathtt{ct}[0]+\mathtt{ct}[1]s$, and to retrieve the component-wise noise from the result. We get

$$
\begin{aligned}
&\mathtt{ct}[0]+\mathtt{ct}[1]s \\
&\quad= \mathtt{ct}'[0]+\left\lceil P^{-1}\langle[\mathtt{ct}'[1]]_{q_{\gamma_0\le i\le\beta}},\mathtt{evk}[0]\rangle\right\rfloor+\left\lceil P^{-1}\langle[\mathtt{ct}'[1]]_{q_{\gamma_0\le i\le\beta}},\mathtt{evk}[1]\rangle\right\rfloor s \\
&\quad= \mathtt{ct}'[0]+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}\mathtt{evk}[0][i]+\tau_0+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}\mathtt{evk}[1]s+\tau_1 s \\
&\quad= \mathtt{ct}'[0]+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}\left[-a_i s+s'P\frac{Q_L}{q_{\gamma_i}}\left[\left(\frac{Q_L}{q_{\gamma_i}}\right)^{-1}\right]_{q_{\gamma_i}}+e_i\right]_{PQ_L} \\
&\qquad+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}[a_i]_{PQ_L}s+\tau_0+\tau_1 s \\
&\quad= \mathtt{ct}'[0]+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}\left[s'\frac{Q_L}{q_{\gamma_i}}\left[\left(\frac{Q_L}{q_{\gamma_i}}\right)^{-1}\right]_{q_{\gamma_i}}+e_i\right]_{PQ_L}+\tau_0+\tau_1 s \\
&\quad= \mathtt{ct}'[0]+\left[\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}\frac{Q_L}{q_{\gamma_i}}\left[\left(\frac{Q_L}{q_{\gamma_i}}\right)^{-1}\right]_{q_{\gamma_i}}+P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}e_i\right]_{PQ_L} \\
&\qquad+\tau_0+\tau_1 s \\
&\quad= \mathtt{ct}'[0]+\mathtt{ct}'[1]s'+[P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}n(\mathtt{evk})[0]]_{PQ_L}+\tau_0+\tau_1 s.
\end{aligned}
$$

From this we can easily see that

$$
n(\mathtt{ct})=n(\mathtt{ct}')+\left(P^{-1}\sum_{i=0}^{\beta}[\mathtt{ct}'[1]]_{q_{\gamma_i}}e_i+\tau_0,\tau_1\right).
$$

We can assume $[\mathtt{ct}'[1]]_{q_{\gamma_i}}$ to be uniformly randomly distributed in the interval $[q_{\gamma_i}]$. Since the terms $e_i=n(\mathtt{evk})[0]$ are drawn from the error distribution χ, they have standard deviation σ_0. We obtain the following.

$$
\sigma_{n(\mathtt{ct})}=\left(\sqrt{\sigma^2_{n(\mathtt{ct}')[0]}+P^{-2}\sum_{i=0}^{\beta}\frac{Nq_{\gamma_i}^2\sigma_0^2}{12P^2}+\frac{1}{12}},\sqrt{\sigma^2_{n(\mathtt{ct}')[1]}+\frac{1}{12}}\right).
$$

□

Lemma 14 (Relinearization). *Let $\{\mathtt{ct},Q_\ell,\Delta\}$ be a ciphertext resulting from* `Tensor`*. Note that $\mathtt{ct}^{(0)}=(\mathtt{ct}[0],\mathtt{ct}^{(0)}[1],\mathtt{ct}^{(0)}[2])$. Let $\{\mathtt{ct},Q_\ell,\Delta\}=$* `Relin`*$(\{\mathtt{ct},Q_\ell,\Delta_0\})=(\mathtt{ct}[0],\mathtt{ct}[1])+$* `KeySwitch`*$(\mathtt{ct}[2],s^2)$. Then, for the coefficient standard deviation of the component-wise noise, we have*

$$
\sigma_{n(\mathtt{Relin}(\mathtt{ct}))}
$$

$$= \left(\sqrt{\sigma^2_{n(\mathtt{ct}[0])} + N\sigma^2_{n(\mathtt{ct}[2])}\sigma^2_{s^2} + \frac{N\sum_{i=0}^{\beta-1} q^2_{\gamma_i}\sigma^2_0}{12P^2} + \frac{1}{12}}, \sqrt{\sigma^2_{n(\mathtt{ct}[1])} + \frac{1}{12}} \right).$$

Proof. Let
$\mathtt{evk}_{s^2\to s} = ((\mathtt{evk}_{s^2\to s}[0][0], \ldots, \mathtt{evk}_{s^2\to s}[\beta-1][0]),$
$(\mathtt{evk}_{s^2\to s}[0][1], \ldots, \mathtt{evk}_{s^2\to s}[\beta-1][1]))$ be the evaluation key, where

$$\begin{aligned}
\mathtt{evk}_{s^2\to s}[i][0] &= \left[a_i s + s^2 P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right)^{-1} \right]_{q_{\gamma_i}} + e_i \right]_{PQ_L} \\
\mathtt{evk}_{s^2\to s}[i][0] &= \left[a_i s + s^2 P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right)^{-1} \right]_{q_{\gamma_i}} + e_i \right]_{PQ_L} \\
\mathtt{evk}_{s^2\to s}[i][1] &= [-a_i]_{PQ_L}.
\end{aligned}$$

By Algorithm 3 in [BMTPH21], the output ciphertext $\mathtt{ct}$ is calculated as

$$\begin{aligned}
(\mathtt{ct}[0], \mathtt{ct}[1]) = (\mathtt{ct}^{(0)}[0] + \lceil P^{-1} \langle ([\mathtt{ct}^{(0)}[2]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_{\beta-1}}}), \\
(\mathtt{evk}_{s^2\to s}[0][0], \ldots, \mathtt{evk}_{s^2\to s}[\beta-1][0]) \rangle \rfloor, \\
\mathtt{ct}^{(0)}[1] + \lceil P^{-1} \langle ([\mathtt{ct}^{(0)}[2]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_{\beta-1}}}), \\
(\mathtt{evk}_{s^2\to s}[0][1], \ldots, \mathtt{evk}_{s^2\to s}[\beta-1][1]) \rangle \rfloor).
\end{aligned}$$

To determine which terms will be cancelled out during decryption, and therefore which terms are part of the component-wise noise, we calculate $\mathtt{Dec}(\mathtt{ct}) - \mathtt{pt}^{(0)}$.

$$\begin{aligned}
&\mathtt{ct}[0] + \mathtt{ct}[1]s - \mathtt{pt}^{(0)} \\
&= \mathtt{ct}^{(0)}[0] + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} \mathtt{evk}_{s^2\to s}[j][0] \right\rfloor \\
&\quad + \mathtt{ct}^{(0)}[1]s + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} \mathtt{evk}_{s^2\to s}[i][1] \right\rfloor - \mathtt{pt}^{(0)} \\
&= \mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s \\
&\quad + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} (\mathtt{evk}_{s^2\to s}[i][0] + \mathtt{evk}_{s^2\to s}[i][1]s) \right\rfloor - \mathtt{pt} \\
&= \mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s \\
&\quad + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} \left(a_i s + s^2 P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right) \right]_{q_{\gamma_i}} + e_i - a_i s \right) \right\rfloor - \mathtt{pt}^{(0)} \\
&= \mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s \\
&\quad + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} \left(s^2 P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_j}} + e_i \right) \right]_{q_{\gamma_i}} \right) \right\rfloor - \mathtt{pt}^{(0)} \\
&= \mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s
\end{aligned}$$

$$
\begin{aligned}
&+ P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} \left(s^2 P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} + e_i \right) \right]_{q_{\gamma_i}} \right) - \mathtt{pt}^{(0)} \\
&+ \tau^{(0)} + \tau^{(1)} s \\
&= \mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1] s + \mathtt{ct}^{(0)}[2] s^2 - \mathtt{pt}^{(0)} \\
&+ P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} e_j + \tau^{(0)} + \tau^{(1)} s.
\end{aligned}
$$

Therefore, for the component-wise noise we have that:

$$
\begin{aligned}
n(\mathtt{ct})[0] &= n(\mathtt{ct}^{(0)})[0] + n(\mathtt{ct}^{(0)})[2] s^2 + \left\lceil P^{-1} \sum_{i=0}^{\beta-1} [\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}} e_i \right\rfloor + \tau^{(0)} \\
&= n(\mathtt{ct}^{(0)})[0] + n_{\mathsf{ks-add}} \\
n(\mathtt{ct})[1] &= n(\mathtt{ct}^{(0)})[1] + \tau^{(1)}.
\end{aligned}
$$

We can assume $[\mathtt{ct}^{(0)}[2]]_{q_{\gamma_i}}$ to be uniformly randomly distributed in $\mathcal{R}_{q_{\gamma_i}}$. The terms therefore have coefficient standard deviation $\frac{q_{\gamma_i}^2}{12}$. Then, by Lemma 1, for the coefficient standard deviation of the component-wise noise, we have the following.

$$
\begin{aligned}
\sigma_{n(\mathtt{ct})} &= \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + N \sigma^2_{n(\mathtt{ct}^{(0)})} \sigma^2_{s^2} + \frac{N \sum_{i=0}^{\beta-1} q_{\gamma_i}^2 \sigma_0^2}{12 P^2} + \frac{1}{12}}, \sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[1]} + \frac{1}{12}} \right) \\
&= \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \sigma^2_{\mathsf{ks-add}(\beta,P)}}, \sqrt{\sigma^2_{n(\mathtt{ct}')[1]} + \frac{1}{12}} \right).
\end{aligned}
$$

□

Lemma 15 (Rotations). *Let $\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}$ be a ciphertext encrypting $\mathtt{pt}^{(0)}$ with respect to a secret key $\mathtt{sk} = (1, s)$. Let $\{\mathtt{ct}, Q_{\ell_0}, \Delta_0\} = \mathtt{Rotate}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, k)$ be a ciphertext encrypting $\Phi_k(m_{\mathtt{ct}})$ with respect to $\Phi_k(\mathtt{sk}) = (1, \Phi_k(s))$. Then we have for the coefficient standard deviation of the component-wise noise*

$$
\sigma_{\mathtt{rot}(\mathtt{ct})} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}[0])} + \sigma^2_{\mathsf{ks-add}}}, \sqrt{\sigma_{n(\mathtt{ct}[1])} + \tfrac{1}{12}} \right).
$$

Proof. The proof is very similar to the proof of Lemma 13. We have as evaluation keys $\mathtt{evk}_{s \to \Phi_k^{-1}(s)} = ((\mathtt{evk}_{s \to \Phi_k^{-1}(s)}[0][0], \ldots, \mathtt{evk}_{s \to \Phi_k^{-1}(s)}[\beta][0]),$
$(\mathtt{evk}_{s \to \Phi_k^{-1}(s)}[0][1], \ldots, \mathtt{evk}_{s \to \Phi_k^{-1}(s)}[\beta][1]))$, where

$$
\begin{aligned}
\mathtt{evk}_{s \to \Phi_k^{-1}(s)}[i][0] &= \left\lceil a_i \Phi_k^{-1}(s) + s P \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right)^{-1} \right]_{q_{\gamma_i}} + e_i \right\rfloor_{PQ_L} \\
\mathtt{evk}_{s \to \Phi_k^{-1}(s)}[i][1] &= [-a_i]_{PQ_L}.
\end{aligned}
$$

By Algorithm 4 in [BMTPH21] the ciphertext after rotation is then calculated as

$$\begin{aligned}\mathtt{ct} =&(\Phi_k(\mathtt{ct}^{(0)}[0] + \lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][0])\rangle\rfloor),\\ &\phi_k(\lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}}),\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][1])\rangle\rfloor)).\end{aligned}$$

And therefore,

$$\begin{aligned}&\mathtt{ct}[0] + \mathtt{ct}[1]s - \Phi_k(\mathtt{pt}^{(0)})\\ &= \Phi_k(\mathtt{ct}^{(0)}[0] + \lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}})\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][0])\rangle\rfloor)\\ &\quad + \Phi_k(\lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}}),\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][1])\rangle\rfloor))s - \Phi_k(\mathtt{pt}^{(0)})\\ &= \Phi_k(\mathtt{ct}^{(0)}[0] + \lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}})\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][0])\rangle\rfloor\\ &\quad + \lceil P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}}),\\ &(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][1])\rangle\rfloor\Phi_k^{-1}(s) - \mathtt{pt}^{(0)})\\ &= \Phi_k\Bigg(\mathtt{ct}^{(0)}[0] + P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}\\ &\Big(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[j][0] + \mathtt{evk}_{s\to\Phi_k^{-1}(s)}[j][1]\Phi^{-1}(s)\Big)\\ &\quad - \mathtt{pt}^{(0)} + \tau^{(0)} + \tau^{(1)}s\Bigg)\\ &= \Phi_k\Bigg(\mathtt{ct}^{(0)}[0] + P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}\\ &\left(sP\frac{Q_L}{q_{\gamma_j}}\left[\left(\frac{Q_L}{q_{\gamma_j}}\right)^{-1}\right]_{q_{\gamma_j}} + e_j\right) + \tau^{(0)} + \tau^{(1)}s - \mathtt{pt}^{(0)}\Bigg)\\ &= \Phi_k\Bigg(\mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s - \mathtt{pt}^{(0)} + P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}e_j\\ &\quad + \tau^{(0)} + \tau^{(1)}s\Bigg)\\ &= \Phi_k(\mathtt{ct}^{(0)}[0] + \mathtt{ct}^{(0)}[1]s - \mathtt{pt}^{(0)}) + \Phi_k(P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}e_j)\mathtt{ct}\\ &\quad + \Phi_k(\tau^{(0)} + \tau^{(1)}s).\end{aligned}$$

Applying $\Phi_{(k_0,k_1)} : X^i \to X^{i(-1)^{k_0}5^{k_1}} \mod (X^N+1)$ permutes the coefficients with at most a sign change since $X^N \equiv -1 \mod (X^N+1)$. Although the structure of the ciphertext is changed, the standard deviation of the noise remains the same. Therefore, the standard deviations of the component-wise noise are the same as in the case of key switching. □

Lemma 16 (Conjugations). *Let $\mathtt{ct}^{(0)}$ be a ciphertext encrypting $\mathtt{pt}^{(0)}$, which is an encoding of m_0 and let $\mathtt{ct} = \mathtt{Conjugation}(\mathtt{ct}^{(0)})$ be the ciphertext encrypting $\mathtt{pt}$, encoding $\overline{m_0}$. Then we have for the coefficient standard deviation of the component-wise noise*

$$\sigma_{\mathtt{conj(ct)}} = \left(\sqrt{\sigma^2_{n(\mathtt{ct}[0])_i} + \sigma^2_{\mathsf{ks-add}}}, \sqrt{\sigma_{n(\mathtt{ct}[1])} + \tfrac{1}{12}}\right).$$

Proof. Conjugation of a ciphertext is calculated by rotating the ciphertext by one slot. Therefore the component-wise noise is the same as that of a rotation by -1 slots. □

B Noise Standard Deviations for Bootstrapping

B.1 RNS-CKKS Bootstrapping

Here we present a high-level overview for CKKS bootstrapping ([CCS19],[CHK+18],[HK20],[LLK+22]) and its RNS variant ([BMTPH21],[LLK+22]).

The steps of an (RNS-)CKKS bootstrapping can be summarized as follows. Let $\mathtt{ct}$ be an RNS-CKKS ciphertext at the lowest level $\ell = 0$, encrypting a plaintext $\mathtt{pt}$.

1. **Mod Raise** In a first step the ciphertext is raised to a higher modulus Q_L, through applying the map $\mathcal{R}_{q_0} \to \mathcal{R}_{q_0} \times \mathcal{R}_{q_1} \times \ldots \times \mathcal{R}_{q_L}$. This map does not apply any scaling to the ciphertext coefficients, but acts as an implicit decryption. Since this decryption is done modulo Q_L and not $Q_0 = q_0$, the new ciphertext $\mathtt{ct}'$ will now encrypt $\mathtt{pt} + Iq_0$ where I is a small norm polynomial representing the q_0 overflows that were not removed. It will be the purpose of the subsequent 3 steps to evaluate a reduction modulo q_0 on $\mathtt{ct}'$.

2. **Coefficients to Slots** The new message $\mathtt{pt} + Iq_0$ is in the coefficient format, thus does not support slot-wise arithmetic. The message $\mathtt{pt} + Iq_0$ is therefore homomorphically transformed from the coefficients to the slots format to allow for slot-wise evaluation of the modulus reduction. The coefficients to slots algorithm is the same as the one used to encode a message from the message space into the plaintext space (up to a bit-reversal permutation), only that it is now evaluated homomorphically. This step returns $\mathsf{DFT}^{-1}(\mathtt{pt} + Iq_0)$.

3. **Evaluate the Sine function** The reduction modulo q_0 is not a polynomial function and cannot be easily evaluated homomorphically. It is approximated by a scaled sine function, which in turn is approximated by its polynomial interpolation. This step homomorphically evaluates the polynomial interpolation of the scaled sine function on $\mathtt{ct}'$, to obtain a ciphertext $\mathtt{ct}''$ encrypting only $\mathsf{DFT}^{-1}(\mathtt{pt})$.

4. **Slots to coefficients** The ciphertext $\mathtt{ct}''$ is transformed back into the coefficients format by homomorphically applying the decoding procedure, returning an encryption of $\mathtt{pt}$ but at level $\ell > 0$.

The steps 2,3, and 4 all add to the noise growth in the ciphertext and consume levels, which is why the resulting bootstrapped ciphertext $\mathtt{ct}''$ may have slightly higher noise than $\mathtt{ct}$, and will be with respect to a level $L - k$ where k is the depth of the bootstrapping circuit. The precise growth of the noise is the subject of this work.

B.2 Noise Standard Deviation for Homomorphic En-and Decoding

In this section, we show how to derive expressions for the coefficient standard deviation of the component-wise noise after evaluation of the bootstrapping circuit, from the estimates

given in Section 3. We first give expressions for the coefficient standard deviation of the component-wise noise after a matrix-ciphertext product. This allows us to give expressions for the standard deviation after homomorphic encoding and decoding. We focus on the improved matrix-ciphertext product as it has been presented in Algorithm 6 in [BMTPH21], stated again in Algorithm 3 in Appendix C since this is the algorithm that is implemented in Lattigo.

Lemma 18 (Matrix-ciphertext Product)**.** *Let $M \in \mathbb{C}^{n\times n}$ be an $n \times n$ matrix and M^i_{diag} be its $i-$th encoded diagonal. Let $\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}$ be a ciphertext. Let $\{\mathtt{ct}_{pre\text{-}rot(i)}, Q_{\ell_0}, P\Delta_0\}$ be the ciphertext resulting from the pre-rotation step in lines $1-6$ of Algorithm 6 in [BMTPH21]. Let $\{\mathtt{ct}_{n_1-loop,j}, Q_{\ell_0}, P\Delta_0\}$ be the $j-$th intermediary result of the summing step in lines $9-12$ of Algorithm 3. Let $\{\mathtt{ct}_{n_2-loop}, Q_{\ell_0}, \Delta_0\}$ be the ciphertext resulting from the rotations in lines $7-17$ in Algorithm 6 in [BMTPH21]. Finally, let $\{\mathtt{ct}, Q_\ell, \Delta\} = \{\mathsf{MCtxt}(\mathtt{ct}^{(0)}, M)0\mathtt{ct}^{(0)} \times M, Q_{\ell_0-1}, \Delta\}$ be the result of the homomorphic evaluation of the ciphertext-matrix product. Choose n_1, n_2 such that $n_1 n_2 = N$. Then the component-wise noise of $\mathtt{ct}$ is built up as follows.*

$$
\begin{aligned}
\sigma_{n(\mathtt{ct}_{pre\text{-}rot(0)})} =& P(\sigma_{n(\mathtt{ct}^{(0)})[0]}, \sigma_{n(\mathtt{ct}^{(0)})[1]}) \\
\sigma_{n(\mathtt{ct}_{pre\text{-}rot(i\geq 1)})} =& P\left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})[0]} + \frac{N\sum_{k=0}^{\beta-1} q_{\gamma_k}\sigma_0^2}{12P^2}}, \sigma_{n(\mathtt{ct}^{(0)})[1]}\right) \\
\sigma_{n(\mathtt{ct}_{n_1-loop,j})} =& \left(\sqrt{\sum_{i=0}^{n_1-1} \sigma^2_{n(\mathtt{ct}_{pre\text{-}rot(i)})[0]}\left(M^{n_1 j+i}_{diag} + \frac{1}{12}\right)}, \right. \\
& \left. \sqrt{\sum_{i=0}^{n_1-1} \sigma^2_{n(\mathtt{ct}_{pre\text{-}rot(i)})[1]}\left(M^{n_1 j+i}_{diag} + \frac{1}{12}\right)}\right) \\
\sigma_{n(\mathtt{ct}_{n_2-loop})} =& \left(\sqrt{\sum_{j=0}^{n_2-1} \sigma^2_{n(\mathtt{Rotate}(\mathtt{ct}_{n_1-loop,j}, n_1 j))[0]}}, \sqrt{\sum_{j=0}^{n_2-1} \sigma^2_{n(\mathtt{Rotate}(\mathtt{ct}_{n_1-loop,j}, n_1 j))[1]}}\right) \\
\sigma_{n(\mathtt{ct})} =& \left(\sqrt{q_{\ell_0}^{-2}\left(P^{-2}\sigma^2_{n(\mathtt{ct}_{n_2-\mathsf{loop})[0]}} + \frac{1}{12}\right) + \frac{1}{12}}, \right. \\
& \left. \sqrt{q_{\ell_0}^{-2}\left(P^{-2}\sigma^2_{n(\mathtt{ct}_{n_2-\mathsf{loop})[1]}} + \frac{1}{12}\right) + \frac{1}{12}}\right)
\end{aligned}
$$

Proof. We have $\mathtt{ct}_{\text{pre-rot}(0)} = P\mathtt{ct}^{(0)}$, the statement about the component-wise noise follows. Furthermore, we have

$$
\begin{aligned}
&\mathtt{ct}_{\mathsf{pre-rot}(i\geq 1)}[0] \\
&= \Phi_i(P\mathtt{ct}^{(0)}[0] + \langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}), \\
&(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle) \\
&= P\Phi_i(\mathtt{ct}^{(0)}[0] + P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}), \\
&(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle) \\
&\mathtt{ct}_{\mathsf{pre-rot}(i\geq 1)}[1]
\end{aligned}
$$

$$
\begin{aligned}
&= \Phi_i(\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),\\
&(\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle)\\
&= P\Phi_i(P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),\\
&(\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle).
\end{aligned}
$$

The summing step in lines $9-12$ of the algorithm sums the pre-rotations for all i and multiplies it with the corresponding diagonal of M. Let $M_{\mathsf{diag}}^{n_1 j+i}$ be the $(n_1 j+i)-$th diagonal. Since the coefficients of M are deterministic, we do not need to consider a random rounding error. Therefore, the component-wise noise of $\mathtt{ct}_{n_1-\mathsf{loop},j}$ is the sum of the component-wise noises of $\mathtt{ct}_{\mathsf{pre}-\mathsf{rot}(i)}$ multiplied by $M^{n_1 j+i}$. That is, we get

$$
\sigma_{n(\mathtt{ct}_{n_1-\mathsf{loop}})} = \Bigg(\sqrt{\sum_{i=0}^{n_1-1} \sigma^2_{n(\mathtt{ct}_{\mathsf{pre}-\mathsf{rot}(i)})[0]} M_{\mathsf{diag}}^{n_1 j+i}},\ \sqrt{\sum_{i=0}^{n_1-1} \sigma^2_{n(\mathtt{ct}_{\mathsf{pre}-\mathsf{rot}(i)})[1]} M_{\mathsf{diag}}^{n_1 j+i}}\Bigg).
$$

After the n_2-loop we obain:

$$
\begin{aligned}
\mathtt{ct}_{n_2-\mathsf{loop}}[0] &= \sum_{j=0}^{n_2-1} \Phi_{n_1 j}(\mathtt{ct}_{n_1-\mathsf{loop},j}[0] + \langle([\lfloor \lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_0}},\\
&\quad \ldots, [\lfloor \lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_{\beta-1}}}), (\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle),\\
&= \sum_{j=0}^{n_2-1} \Phi_{n_1 j}(\mathtt{ct}_{n_1-\mathsf{loop},j}[0] + \lceil P^{-1}\langle([\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_0}},\\
&\quad \ldots, [\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_\beta}})(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle\rfloor),\\
&= \sum_{j=0}^{n_2-1} \mathtt{Rotate}(\mathtt{ct}_{n_1-\mathsf{loop},j})[0].
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{ct}_{n_2-\mathsf{loop}}[1] &= \sum_{j=0}^{n_2-1} \Phi_{n_1 j}(\langle([\lfloor \lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_0}}, \ldots, [\lfloor \lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_\beta}}),\\
&\quad (\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle),\\
&= \sum_{j=0}^{n_2-1} \Phi_{n_1 j}(\lceil P^{-1}\langle([\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_\beta}}),\\
&\quad (\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle\rfloor),\\
&= \sum_{j=0}^{n_2-1} \mathtt{Rotate}(\mathtt{ct}_{n_1-\mathsf{loop},j})[1].
\end{aligned}
$$

Therefore, the standard deviation of the component-wise noise of $\mathtt{ct}_{n_2-\mathsf{loop}}$ is just a sum of n_2 rotations. The stated results thus follow from Lemmas 9 and 15. $\mathtt{ct}$ is then calculated as $\mathtt{Rescale}(\lceil P^{-1}\mathtt{ct}_{n_2-\mathsf{loop}}\rfloor)$. Thus we have

$$\begin{aligned} n(\mathtt{ct})[i] &= q_{\ell_0}^{-1} n(\lceil P^{-1} n(\mathtt{ct}_{n_2-\mathsf{loop}}\rfloor)[i]) + \tau^{(i_0)} \\ &= q_{\ell_0}^{-1}(P^{-1} n(\mathtt{ct}_{n_2-\mathsf{loop}})[i] + \tau^{(i_1)}) + \tau^{(i_1)}, \end{aligned}$$

where $i \in \{0, 1\}$. The stated result for the standard deviation follows. □

From these elements we can then build up the noise estimates for the Coefficients to Slots transformation `CoeffToSlots`, and the Slots to Coefficients transformation `SlotsToCoeff`.

Combining the results from Section 3 with Lemma 18, we can get a noise estimate for the homomorphic encoding and decoding operations `Coefficients to Slots` and `Slots to coefficients`. We give the noise estimates based on Algorithm 4 and 1 as stated in Appendix C.

Lemma 19 (Coefficients to Slots)**.** *Let* $\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}$ *be a ciphertext encrypting a plaintext* $\mathtt{pt}$ *and let* $(\{\mathtt{ct}^{(1)}, Q_{\ell-k}, \Delta\}, \{\mathtt{ct}^{(2)}, Q_{\ell-k}, \Delta\}) = \mathtt{CtS}(\{\mathtt{ct}^{(0)}, Q_\ell, \Delta\})$ *be its transformation from coefficient representation into slots representation. The coefficients to slots representation outputs two ciphertexts on one input. Let* SF^{-1} *be the inverse of the Special Fourier Transform. Let* $\frac{1}{2}SF^{-1} = \prod_{i=0}^{k-1} \left(\frac{1}{2}\right)^{\frac{1}{k}} SF_i^{-1}$*, a decomposition of the matrix* SF^{-1} *into sub-matrices. Let* $E_i := \left\lceil \Delta\tau^{-1}\left(\left(\frac{1}{2}\right)^{\frac{1}{k}} SF_i^{-1}\right)\right\rfloor$ *be the diagonal-wise encoding of the* $\left(\frac{1}{2}\right)^{\frac{1}{k}} SF_i^{-1}$*. Then following Algorithm 4, we have for the coefficient standard deviation of the component-wise noise*

$$\begin{aligned} \sigma_{n(\mathtt{ct}^{(1)})} =& \left(\sqrt{2\left(\sigma^2_{n(\mathit{MCtxt}(\ldots,(\mathit{MCtxt}(\mathtt{ct}^{(0)},E_{k-1}),E_{k-2}),\ldots,E_0))[0]} + n_{\mathsf{ks-add}(\beta,P,\mathtt{ct}^{(0)}[1],s)}\right)} \right. \\ & \left., \sqrt{2\left(\sigma^2_{n(\mathit{MCtxt}(\ldots,(\mathit{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \frac{1}{12}\right)} \right) \\ \sigma_{n(\mathtt{ct}^{(2)})} =& \left(\sqrt{2\left(\sigma^2_{n(\mathit{MCtxt}(\ldots,(\mathit{MCtxt}(\mathtt{ct}^{(0)},E_{k-1}),E_{k-2}),\ldots,E_0))[0]} + n_{\mathsf{ks-add}(\beta,P,\mathtt{ct}^{(0)}[1],s)}\right)} \right. \\ & \left., \sqrt{2\left(\sigma^2_{n(\mathit{MCtxt}(\ldots,(\mathit{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \frac{1}{12}\right)} \right). \end{aligned}$$

Remark 3. The Special Fourier matrices are decomposed for better efficiency. A description of the decomposition can be found in [CHH18].

Proof. The coefficients to slots operation is calculated following [BMTPH21] as follows

$$\begin{aligned} \mathtt{ct}^{(1)} &= (E \times \mathtt{ct} + \overline{E \times \mathtt{ct}}) \\ &= (E_0 \times \ldots \times E_k \times \mathtt{ct} + \overline{E_0 \times \ldots \times E_k \times \mathtt{ct}}) \\ \mathtt{ct}^{(2)} &= -i(E \times \mathtt{ct} - \overline{E \times \mathtt{ct}}) \\ &= -i(E_0 \times \ldots \times E_k \times \mathtt{ct} + \overline{E_0 \times \ldots \times E_k \times \mathtt{ct}}). \end{aligned}$$

That is we have

$$\begin{aligned}
\mathtt{ct}^{(1)} =& \mathtt{Add}(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0),\\
&\mathtt{Conjugate}(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0)))\\
\mathtt{ct}^{(2)} =& \mathtt{MultConst}\Big(\mathtt{Add}(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0),\\
&\mathtt{Conjugate}(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))),-i\Big)
\end{aligned}$$

By Lemmas 9, 11, 16, and 18. Thus we get

$$\begin{aligned}
\sigma_{n(\mathtt{ct}^{(0)})[0]} =& \Big(\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]}\\
&\quad + \sigma^2_{n(\mathtt{Conjugate}(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0)))[0]}\Big)^{\frac{1}{2}}\\
=& \Big(\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]}\\
&\quad + \sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \sigma^2_{\mathsf{ks-add}[0]}\Big)^{\frac{1}{2}}\\
=& \Big(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \sigma_{\mathsf{ks-add}[0]}\Big)^{\frac{1}{2}}\\
\sigma_{n(\mathtt{ct}^{(0)})[1]} =& \Big(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \frac{1}{12}\Big)^{\frac{1}{2}}
\end{aligned}$$

$$\begin{aligned}
\sigma_{n(\mathtt{ct}^{(2)})[0]} =& \Big(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))_{[}0]} + \sigma^2_{\mathsf{ks-add}[0]]}\Big)^{\frac{1}{2}}\\
\sigma_{n(\mathtt{ct}^{(2)})[1]} =& \Big(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[1]} + \frac{1}{12}\Big)^{\frac{1}{2}}.
\end{aligned}$$

□

Algorithm 1 Slots To Coefficients

Require: $\mathtt{ct}^{(0)}, \mathtt{ct}^{(1)}$, $\mathsf{SF} = \prod_{i=0}^{k-1} \mathsf{SF}_i$.
1: $\mathtt{ct} \leftarrow \mathtt{ct}^{(0)} + i \cdot \mathtt{ct}^{(1)}$
2: **for** $i \leftarrow 0$ **to** $k-1$ **do**
3: $\quad \mathtt{ct} \leftarrow \mathtt{MatMul}(\mathtt{ct}, \mathsf{SF}_i)$
4: $\quad \mathtt{ct} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
5: **end for**
6: **return** ct

Lemma 20 (Slots to Coefficients)**.** *Let* $(\{\mathtt{ct}^{(1)}, Q_\ell, \Delta\}, \{\mathtt{ct}^{(2)}, Q_\ell, \Delta\})$ *be two ciphertexts in slots format, each encoding* n *real coefficients. Let* $\{\mathtt{ct}^{(0)}, Q_{\ell-k}, \Delta_0\} = \mathtt{StC}(\{\mathtt{ct}^{(1)}, Q_\ell, \Delta\}, \{\mathtt{ct}^{(2)}, Q_\ell, \Delta\})$ *be the ciphertext that has been transformed from Slots to Coefficient format. Let* SF *be the special Fourier transform matrix and let* $SF = \prod_{i=0}^{k-1} SF_i$

be its decomposition. Let $E_i \leftarrow \lceil \Delta\tau^{-1}(SF_i) \rfloor$ *be its encoding. Then we have for the coefficient standard deviation of the component-wise noise*

$$\sigma_{n(\mathtt{ct}^{(1)})} = \left(\sqrt{\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)}+\mathtt{ct}^{(1)},E_{k-1}),E_{k-2}),\ldots,E_0))[0]}} \right.$$
$$\left. , \sqrt{\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)}+\mathtt{ct}^{(1)},E_{k-1}),E_{k-2}),\ldots,E_0))[1]}} \right).$$

Proof. Following [BMTPH21], the Slots to Coefficient operation is evaluated as

$$\begin{aligned}\mathtt{ct} =& \mathtt{Add}(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(1)}, E_0), \ldots), E_k),\\ & \mathtt{MultConst}(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(2)}, E_0), \ldots), E_k), i)).\end{aligned}$$

W get for the coefficient variance of the component-wise noise

$$\begin{aligned}\sigma^2_{n(\mathtt{ct})} &= \sigma^2_{n(\mathtt{A}(\mathtt{MC}(\ldots(\mathtt{MC}(\mathtt{ct}^{(1)},E_0),\ldots),E_k),\mathtt{MCons}(\mathtt{MC}(\ldots(\mathtt{MC}(\mathtt{ct}^{(2)},E_0),\ldots),E_k),i)))}\\ &= \sigma^2_{n(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(1)},E_0),\ldots),E_k))}\\ &\quad + \sigma^2_{n(\mathtt{MultConst}(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(2)},E_0),\ldots),E_k),i))},\end{aligned}$$

and further

$$\begin{aligned}\sigma^2_{n(\mathtt{ct})} =& \sigma^2_{n(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(1)},E_0),\ldots),E_k))}\\ &+ \sigma^2_{n(\mathtt{MCtxt}(\ldots(\mathtt{MCtxt}(\mathtt{ct}^{(2)},E_0),\ldots),E_k),i)}.\end{aligned}$$

The Lemma follows directly from this. □

B.3 Noise Standard Deviation for Polynomial Evaluation

We proceed by giving expressions for the standard deviation that build up to a noise estimate of the `EvalMod1` Algorithm 6, which is the third step in the bootstrapping circuit. The `EvalMod1` algorithm consists of approximating the function $f_{\mathsf{mod1}} : x \mod 1$. Many works have proposed solutions for this step [CHK+18, CCS19, LLK+22, LLL+21, LLKN20, JM20, JM22, KPK+22]. We choose the approach of [HK20], as it provides in practice the best overall performance in term of precision and depth consumption and is the default approach in Lattigo. It approximates f_{mod1} with a phase-shifted scaled cosine function, which enables an HE friendly evaluation of the double angle formula. The scaled cosine function is itself approximated through polynomial interpolation using a powerful basis consisting of Chebyshev polynomials. Afterwards, it calls the `EvaluatePolynomial` algorithm, which evaluates the polynomial that has been interpolated to approximate the f_{mod1} function. In this section, we give expressions for the standard deviation after polynomial evaluation. We use the `EvaluatePolynomial` algorithm currently implemented in Lattigo [Lat24] for our analysis and we recall it in Algorithm 7 in Appendix C.

We first give the noise estimates for the power basis. Chebyshev polynomials can be defined via the following recursion

$$\begin{aligned} T_0(t) &:= 1 \\ T_1(t) &:= t \\ T_{n+1}(t) &:= 2T_1(t)T_n(t) - T_{n-1}(t). \end{aligned}$$

However, we will use the following recursion: $T_{j=a+b} = 2T_a(t)T_b(t) + T_{|a-b|}(t)$, where $a = 2^{\lfloor \log(i) \rfloor} - 1$ and $b = j + 1 - 2^{\lfloor \log(i) \rfloor}$ if j is not a power of two, else $a = b = j/2$. While these two recursions are mathematically equivalent, the second one enjoys a better noise growth and is depth-optimal when evaluated homomorphically, as is pointed out in [HK20].

We give the noise estimates recursively and not in a closed form for the following reason: in FHE, the order in which operations are applied matters; $(x + y)z$ and $xz + yz$ may have different noise growth, even though encrypt the same underlying plaintext. Since the computation of the Chebyshev polynomials is mostly implemented recursively in practice, giving the noise estimates non-recursively would lead to an inaccurate modelling of the noise growth.

Lemma 21 (Power Basis)**.** *Let $T_j(t)$ be the $j-$th Chebyshev polynomial, defined via the following recursion.*

$$\begin{aligned} T_0(t) &= 1 \\ T_1(t) &= t \\ T_{j=a+b}(t) &= 2T_a(t)T_b(t) + T_{|a-b|}(t). \end{aligned}$$

Then we have for the coefficient standard deviation of the component-wise noise

$$\begin{aligned} \sigma_{n(T_0(t))} =& (0,0) \\ \sigma_{n(T_1(t))} =& \sigma_{n(t)_i} \\ \sigma_{n(T_{a+b}(t))} =& \Bigg(\Bigg(q_{\tilde{\ell}}^{-2} \Bigg(4\sigma^2_{n(\mathtt{Tensor}(T_a(t),T_b(t))[0])_i} + \sigma^2_{\mathsf{ks-add}} \\ & + \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor \sigma^2_{n(T_{|a-b|}(t)[0])_i} + \frac{1}{12} \Bigg) + \frac{1}{12} \Bigg)^{\frac{1}{2}}, \\ & \Bigg(q_{\tilde{\ell}}^{-2} \Bigg(4\sigma^2_{n(\mathtt{Tensor}(T_a(t),T_b(t))[1])} + \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor \sigma^2_{n(T_{|a-b|}(t)[1])} + \frac{1}{3} \Bigg) \\ & + \frac{1}{12} \Bigg)^{\frac{1}{2}} \Bigg), \end{aligned}$$

where $q_{\tilde{\ell}}$ is the top most factor of the decomposition of $Q_{\tilde{\ell}} = \min(Q_{T_a(t)}, Q_{T_b(t)})$ and $\Delta_a, \Delta_b, \Delta_{|a-b|}$ the scaling factors of the corresponding Chebyshev polynomials.

Proof. The first part of the proof is trivial: $T_0(t) = 1$, therefore it is deterministic and does not have a noise standard deviation. $T_1(t) = t$, thus the component-wise noise of $T_1(t)$ is the same as of t.

$T_i(t)$ is calculated from a tensor product, followed by a relinearization and a rescaling, a multiplication by constant and an addition. We therefore get for the component-wise noise of $T_i(t)$

$$n(T_{a+b}(t))[0] = n(\mathtt{Rescale}(\mathtt{Add}(\mathtt{MultConst}(\mathtt{Relin}(\mathtt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\},$$

$$
\begin{aligned}
&\{T_b, Q_{T_b(t)}, \Delta_b\})), 2), \{T_{|a-b|(t)}, Q_{T_{|a-b|}(t)}, \Delta_{|a-b|}\}))))[0] \\
=&q_{\tilde{\ell}}^{-1} n(\texttt{Add}(\texttt{MultConst}(\texttt{Relin}(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \\
&\{T_b, Q_{T_b(t)}, \Delta_b\})), 2), \{T_{|a-b|(t)}, Q_{T_{|a-b|}(t)}, \Delta_{|a-b|}\}))))[0] + \tau_0 \\
=&q_{\tilde{\ell}}^{-1} (n(\texttt{MultConst}(\texttt{Relin}(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \\
&\{T_b, Q_{T_b(t)}, \Delta_b\})), 2))[0] + \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor n(T_{|a-b|(t)})[0] + \tau_1) + \tau_0 \\
=&q_{\tilde{\ell}}^{-1} (2n(\texttt{Relin}(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \{T_b, Q_{T_b(t)}, \Delta_b\})))[0] \\
&+ \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor n(T_{|a-b|(t)})[0] + \tau_1) + \tau_0 \\
=&q_{\tilde{\ell}}^{-1} (2n(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \{T_b, Q_{T_b(t)}, \Delta_b\}))[0] \\
&+ 2n_{\mathsf{relin-add}} + \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor n(T_{|a-b|(t)})[0] + \tau_1) + \tau_0.
\end{aligned}
$$

Similarly, we get for $n(T_{a+b}(t))[1]$

$$
\begin{aligned}
n(T_{a+b}(t))[1] =&q_{\tilde{\ell}}^{-1} (2n(\texttt{Relin}(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \{T_b, Q_{T_b(t)}, \Delta_b\})))[1] \\
&+ \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor n(T_{|a-b|(t)})[1] + \tau_1) + \tau_0 \\
=&q_{\tilde{\ell}}^{-1} (2n(\texttt{Tensor}(\{T_a, Q_{T_a(t)}, \Delta_a\}, \{T_b, Q_{T_b(t)}, \Delta_b\}))[1] \\
&+ 2\tau_2 + \left\lceil \frac{\Delta_a \Delta_b}{\Delta_{|a-b|}} \right\rfloor n(T_{|a-b|(t)})[0] + \tau_1) + \tau_0.
\end{aligned}
$$

The standard deviations of the component-wise noise follow directly from the above, again modelling the τ_i as being uniform random with variance $\frac{1}{12}$. □

Let $p(t)$ be the a degree d polynomial that we want to evaluate on $\texttt{ct}$, and let $T = \{\{T_0, T_1, \ldots, T_{2^{l-1}}\}, \{T_{2^l}, T_{2^{l+1}}, \ldots, T_{2^m}\}\}$ be a power basis with $m = \lceil \log(d) \rceil$ and $l = \lfloor m/2 \rfloor$. Let c_i be coefficients, such that $p(t) = \sum_{i=0}^{d} c_i T_i(t)$.

Lemma 22 (BabyStep)**.** *Let $p(t) = \sum_{i=0}^{d} c_i T_i(t)$ a polynomial with $c_i \in \mathbb{C}$, $\mathbf{T} = \{1, \texttt{ct}, T_2(\texttt{ct}), \ldots, T_d(\texttt{ct})\}$ a pre-computed power-basis and Δ a target scaling factor and let $\{T_d(\texttt{ct}), Q_\ell, \Delta'\}$. Then we have for the standard deviation of the component-wise after evaluating $p(t)$ on $\texttt{ct}$ using Algorithm 9*

$$
\sigma_{n(\mathit{BabyStep}(p(t),T,\Delta)} = \sqrt{\sum_{j=0}^{d} \sigma^2_{n(\mathit{MultConst}(c'_j, \mathbf{T}_j))}},
$$

where $c'_j = \left\lceil c_j \cdot \frac{\Delta \cdot q_\ell}{\Delta'} \right\rfloor$ where $q_\ell = Q_\ell / Q_{\ell-1}$.

Proof. The proof follows directly from Algorithm 9. The algorithm multiplies the ciphertexts $T_i(\texttt{ct})$ by $c'_j = \left\lceil c_j \frac{\Delta q_\ell}{\Delta'} \right\rfloor$ and adds the results for i from 0 to $d-1$. The noise estimate therefore is a direct consequence of Lemmas 9 and 11. □

Lemma 23 (GiantStep)**.** *Let $\{\texttt{ct}_0, \ell - 1, \Delta_0 = \Delta_1 \Delta_2\}$, $\{\texttt{ct}_1, \ell, \Delta_1 q_\ell\}$ and $\{\texttt{ct}_2, \ell - 1, \Delta_2\}$ be ciphertexts satisfying the input requirements of the $\mathit{GiantStep}$ (Algorithm 10) encrypting $\texttt{pt}_0$, $\texttt{pt}_1$ and $\texttt{pt}_2$ respectively. Then we have for the standard deviation of the component-wise noise after evaluating Algorithm 10*

$$
\sigma^2_{n(\texttt{ct}'_1)} = \sigma^2_{n(\mathit{Rescale}(\mathit{Relinearize}(\texttt{ct}_1)))}
$$

$$= \left(q_\ell^{-2} \left(\sigma^2_{n(ct_1[0])} + \sigma^2_{\textit{ks-add}} \right) + \frac{1}{12}, q_\ell^{-2} \left(\sigma^2_{n(ct_0[1])} + \frac{1}{12} \right) + \frac{1}{12} \right)$$

$$\sigma_{n(\texttt{GiantStep}(ct_0, ct_1, ct_2)}$$
$$= \left(\sqrt{\sigma^2_{n(ct_0[0])} + N \left(\sigma^2_{pt_1} + \sigma^2_{n(ct'_1[0])} \right) \left(\sigma^2_{pt_2} + \sigma^2_{n(ct_2[0])} \right)}, \right.$$
$$\sqrt{\sigma^2_{n(ct_0)[1]} + N \left(\sigma^2_{n(ct_2)[1]} \left(\sigma^2_{pt_1} + \sigma^2_{n(ct_1)[0]} \right) + \sigma^2_{n(ct_1)[1]} \left(\sigma^2_{pt_2} + \sigma^2_{n(ct_2)[0]} \right) \right)},$$
$$\left. \sqrt{N \sigma^2_{n(ct_1)[1]} \sigma^2_{n(ct_2)[1]}} \right)$$

Proof. The proof follows directly from Lemmas 5,9,12,14. □

Lemma 24 (Polynomial Evaluation)**.** *Let* $\{\texttt{ct}, Q_\ell, \Delta\}$ *be a ciphertext, and* $p(t) = \sum_{i=0}^{d} c_i T_i(t)$ *be a polynomial. Let* $p_i(t)$ *be such that* $p(t) = \sum_{i=0}^{2^{m-\ell}-1} p_i(t) T_{i2^\ell}(t)$. *Let* $b_i = \texttt{Babystep}(p_i(t), T, \Delta'_i)$, $b_j^{(0)} = \texttt{GiantStep}(b_{2j}, b_{2j+1}, T_{2^\ell})$, *and* $b_j^{(i)} = \texttt{GiantStep}(b_{2j}^{(i-1)}, b_{2j+1}^{(i-1)}, T_{2^{\ell+i}})$, *for* $0 < i \leq m - \ell - 1$. *Then we get for the coefficient standard deviation of the component-wise noise of* $\{p(\texttt{ct}), Q_{\ell - \lceil \log(d) \rceil}, \Delta'\} = \texttt{EvaluatePolynomial}(\{\texttt{ct}, Q_\ell, \Delta\}, p(t))$, *for* Δ' *a target scale*

$$\sigma_{n(p(\texttt{ct}))} = \sqrt{q^{-2}_{\ell - \lceil \log(d) \rceil} \sigma^2{}_{n(\textit{Relin}(b_0^{m-\ell-1}))} + \left(\frac{1}{12}, \frac{1}{12} \right)},$$

where

$$\sigma_{n(b_j^{(i)})} = \sigma_{n(\texttt{GiantStep}(b_{2j}^{(i-1)}, b_{2j+1}^{(i-1)}, T_{2^{\ell+i}}))}$$
$$\sigma_{n(b_j)^{(0)}} = \sigma_{n(\texttt{GiantStep}(b_{2j}, b_{2j+1}, T_{2^\ell}))}$$
$$\sigma_{n(b_j)} = \sigma_{n(\textit{BabyStep})(p_i(t), T, \Delta'_i)}.$$

Proof. The proof of the noise estimates of $b_j, b_j^{(0)}, b_j^{(i)}, i > 0$ follows trivially from the definitions. Upon finishing iterating through the outer loop, we have $i = m - \ell - 1$, and therefore the inner loop iterates through $j = 0$ to $j = \frac{2^{m-\ell}}{2^{i-1}} - 1 = \frac{2^{m-\ell}}{2^{m-\ell-1+1}} - 1 = 0$. Therefore, the result of the iteration through the two loops is

$$b_0^{(m-\ell-1)} = \texttt{GiantSteps}(b_0^{(m-\ell-2)}, b_1^{(m-\ell-2)}, T_{2^{m-1}}).$$

This result is relinearized and rescaled by $q_{\ell - \lceil \log(d) \rceil}$, since the algorithm consumes depth $\lceil \log(d) \rceil$. This is returned as $p(\texttt{ct})$. Therefore, we obtain

$$\sigma_{n(p(\texttt{ct}))} = \sqrt{q^{-2}_{\ell - \lceil \log(d) \rceil} \sigma^2{}_{n(\texttt{Relin}(b_0^{(m-\ell-1)}))} + \left(\frac{1}{12}, \frac{1}{12} \right)}.$$

□

B.4 Noise Standard Deviation for the Homomorphic Modulus Reduction

Lastly, we give an expression for the standard deviation of the component-wise noise after homomorphic modulus reduction, which is the last building block needed.

Lemma 25 (EvalMod1)**.** *Let* $\{\mathtt{ct}, Q_{\ell+1}, \Delta\}$ *be a ciphertext,* $p(t)$ *a degree* d *Chebyshev implementation of* $\left(\frac{q_0}{2^{\lceil \log(q_0)\rfloor}}\frac{1}{2\pi}\right)^{2^{-r}} \cos\left(2\pi\frac{-0.25}{2^r}\right)$*, and* K *the range of interpolation. Let* $\mathtt{ct}_{\mathsf{final}} = \mathtt{EvalMod1}(\{\mathtt{ct}, Q_{\ell-r}, \Delta\}, p(t), K)$ *be the result of Algorithm 6. Let* $\mathtt{ct}^{(j)}$ *be the result after the* $j-$*th iteration of the loop in lines* $10 - -16$ *in Algorithm 6. Let* $\{\mathtt{ct}', Q_{\ell-\lceil \log(d)\rceil}, \Delta_r\} = \mathtt{EvaluatePolynomial}(\{\mathtt{ct}'', Q_\ell, \Delta\}, p(t))$*, and* $\{\mathtt{ct}'', Q_\ell, \Delta\} = \mathtt{AddConst}(\mathtt{Rescale}(\mathtt{MultConst}(\{\mathtt{ct}, Q_{\ell+1}, \Delta\}, 2)), -\left(\frac{-0.25}{2^r K}\right))$*. Then we have for the coefficient standard deviation of the component-wise noise of* $\mathtt{ct}_{\mathsf{final}}$

$$\sigma_{n(\mathtt{ct}_{\mathsf{final}})} = \sqrt{q_{\ell-r}^{-2}\boldsymbol{\sigma^2}_{n(\mathtt{ct}^{(r-1)})} + \left(\frac{1}{12}, \frac{1}{12}\right)},$$

where

$$\sigma_{n(\mathtt{ct}^{(j)})} = \sqrt{4\boldsymbol{\sigma^2}_{n(\mathtt{Relin}(\mathtt{Tensor}(\mathtt{ct}^{(j-1)}, \mathtt{ct}^{(j-1)})))} + \left(\frac{q_{\ell-1}^{-2}+1}{12}, \frac{q_{\ell-1}^{-2}+1}{12}\right)},$$

for $0 < j \leq r-1$*, and*

$$\begin{aligned}
\sigma_{n(\mathtt{ct}^{(0)})} &= \sqrt{4\boldsymbol{\sigma^2}_{n(\mathtt{Relin}(\mathtt{Tensor}(\mathtt{ct}', \mathtt{ct}')))} + \left(\frac{q_{\ell-1}^{-2}+1}{12}, \frac{q_{\ell-1}^{-2}+1}{12}\right)} \\
\sigma_{n(\mathtt{ct}'')} &= \sigma_{n(\mathtt{EvaluatePolynomial}(\mathtt{ct}'', p))} \\
\sigma_{n(\mathtt{ct}')} &= \sqrt{2^{\frac{2\lceil \log(q_0)\rfloor}{q_0}}\boldsymbol{\sigma^2}_{n(\mathtt{ct})} + \left(\frac{q_\ell^{-2}+1}{12}, \frac{q_\ell^{-2}+1}{12}\right)}.
\end{aligned}$$

Proof. We derive the noise estimates by first calculating the noise estimates for the loops in the `EvalMod` algorithm. Then we combine them into noise estimates for the full algorithm. First we see that

$$\begin{aligned}
n(\mathtt{ct}'') &= n(\mathtt{AddConst}(\mathtt{Rescale}(\mathtt{MultConst}(\mathtt{ct}, 2^{\frac{\lceil \log(q_0)\rfloor}{q_0}})), \frac{-0.25}{2^r K})) \\
&= n(\mathtt{Rescale}(\mathtt{MultConst}(\mathtt{ct}, 2^{\frac{\lceil \log(q_0)\rfloor}{q_0}}))) \\
&= q_\ell^{-1} n(\mathtt{MultConst}(\mathtt{ct}, 2^{\frac{\lceil \log(q_0)\rfloor}{q_0}})) + (\tau_{00}, \tau_{01}) \\
&= q_\ell^{-1}\left(2^{\frac{\lceil \log(q_0)\rfloor}{q_0}} q_\ell n(\mathtt{ct}) + (\tau_{10}, \tau_{11})\right) + (\tau_{00}, \tau_{01}) \\
&= 2^{\frac{\lceil \log(q_0)\rfloor}{q_0}} n(\mathtt{ct}) + q_\ell^{-1}(\tau_{10}, \tau_{11}) + (\tau_{00}, \tau_{01}).
\end{aligned}$$

Thus,

$$\begin{aligned}
\sigma_{n(\mathtt{ct}'')} &= \sqrt{\left(2^{\frac{\lceil \log(q_0)\rfloor}{q_0}}\right)^2 \boldsymbol{\sigma^2}_{n(\mathtt{ct})} + \left(q_\ell^{-2}\frac{1}{12}, q_\ell^{-2}\frac{1}{12}\right) + \left(\frac{1}{12}, \frac{1}{12}\right)} \\
&= \sqrt{2^{\frac{2\lceil \log(q_0)\rfloor}{q_0}}\boldsymbol{\sigma^2}_{n(\mathtt{ct})} + \left(\frac{q_\ell^{-2}+1}{12}, \frac{q_\ell^{-2}+1}{12}\right)}.
\end{aligned}$$

The result for $n(\mathtt{ct}')$ can directly be seen from the definition of $\mathtt{ct}'$. For $n(\mathtt{ct}^{(j)}), 0 < j \leq r-1$ we get

$$
\begin{aligned}
&n(\mathtt{ct}^{(j)}) \\
&= n(\mathtt{Rescale}(\mathtt{AddConst}(\mathtt{MultConst}(\mathtt{Relin}(\mathtt{Tensor}(\mathtt{ct}^{(j-1),\mathtt{ct}^{(j-1)}}))),2), -\left(\frac{1}{2\pi}\right)^2 - r + j)) \\
&= q_{\ell-j}^{-1} n(\mathtt{MultConst}(\mathtt{Relin}(\mathtt{Tensor}(\mathtt{ct}^{(j-1),\mathtt{ct}^{(j-1)}})),2)) + (\tau_{00}, \tau_{01}) \\
&= q_{\ell-j}^{-1}(2q_{\ell-j} n(\mathtt{Relin}(\mathtt{Tensor}(\mathtt{ct}^{(j-1)}, \mathtt{ct}^{(j-1)}))) + (\tau_{10}, \tau_{11})) + (\tau_{00}, \tau_{01}) \\
&= 2n(\mathtt{RelinTensor}(\mathtt{ct}^{(j-1),\mathtt{ct}^{(j-1)}})) + q_{\ell-j}^{-1}(\tau_{10}, \tau_{11}) + (\tau_{00}, \tau_{01}),
\end{aligned}
$$

from which both the standard deviation for $n(\mathtt{ct}^{(j)})$ and for $n(\mathtt{ct}^{(0)})$ easily follow. Finally, we have

$$
\begin{aligned}
n(\mathtt{ct}_{\mathsf{final}}) &= n(\mathtt{Rescale}(\mathtt{ct}^{(r-1)})) \\
&= q_{\ell-r}^{-1} n(\mathtt{ct}^{(r-1)}) + (\tau_{00}, \tau_{01}),
\end{aligned}
$$

from which the result easily follows. □

B.5 Noise Standard Deviation for Bootstrapping

Lemma 26 (Bootstrapping)**.** *Let $\mathtt{ct}$ be bootstrapped ciphertext, and let $\tilde{\mathtt{ct}}$ be the ciphertext before bootstrapping. Then we can give the component-wise noise of $\mathtt{ct}$ as*

$$n(\mathtt{ct}) = n(\mathit{StC}(\mathit{EvalSine}(\mathit{CtS}(\tilde{\mathtt{ct}})))).$$

Proof. The proof is trivial. □

C Algorithms

Algorithm 2 Optimized Hoisting Rotations (Algorithm 4 in [BMTPH21])

Require: $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) \in \mathcal{R}_{Q_\ell}^2$ and a set of rotation keys $\mathsf{evk}_{s \to \Phi_k^{-1}(s)}$.
Ensure: $\mathbf{v}$ a list containing each k rotation of $\mathtt{ct}$.

1: $\mathbf{d} \leftarrow [[\mathtt{ct}[1]_{q_{\gamma_0 \leq i < \beta}}]]_{PQ_\ell}$ ▷ Decompose
2: **for** all k **do**
3: $\quad (a, b) \leftarrow (\langle \mathbf{d}, \mathsf{evk}^{(0)}_{s \to \Phi_k^{-1}(s)} \rangle, \langle \mathbf{d}, \mathsf{evk}^{(1)}_{s \to \Phi_k^{-1}(s)} \rangle)$
4: $\quad (a, b) \leftarrow (\lceil P^{-1} a \rfloor \lceil P^{-1} b \rfloor)$
5: $\quad \mathbf{v}_k \leftarrow (\Phi_k(\mathtt{ct}[0] + a, \Phi_k(b)))$
6: **end for**
7: **return** $\mathbf{v}$

Algorithm 3 Double-hoisting BSGS matrix-ciphertext algorithm (Algorithm 6 in [BMTPH21])

Require: $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) \in \mathcal{R}^2_{Q_i}$, $M_{\mathsf{diag}} \in \mathcal{R}_{\mathcal{PQ}_\rangle}$ the pre-rotated diagonals of $M_{n\times n}$, where $n = n_1 n_2$. $(\mathsf{rot}^0_{q_{\gamma_i},\Phi_k}, \mathsf{rot}^1_{q_{\gamma_i},\Phi_k})$ set of necessary rotation keys.

Ensure: $\mathtt{ct}' = M \times \mathtt{ct}$

1: $\mathbf{d} \leftarrow [[\mathtt{ct}[1]_{q_{\gamma_i}}]_{0\leq i<\beta}]_{PQ_i}$
2: $(a^{(0)}, b^{(0)}) \leftarrow (P\mathtt{ct}[0], P\mathtt{ct}[1])$
3: **for** $i = 1; i < n_1; i++$ **do**
4: $\quad a^{(i)} \leftarrow \Phi_i(a^{(0)} + \langle \mathbf{d}, \mathsf{rot}^0_{q_{\gamma_i},\Phi_k}\rangle)$
5: $\quad b^{(i)} \leftarrow \Phi_i(\langle \mathbf{d}, \mathsf{rot}^1_{q_{\gamma_i},\Phi_k}\rangle)$
6: **end for**
7: $(\mathtt{ct}[0], \mathtt{ct}[1]) \leftarrow (0, 0)$
8: **for** $j = 0; j < n_2; j++$ **do**
9: $\quad (u^{(0)}, u^{(1)}) \leftarrow (0, 0)$
10: $\quad$ **for** $i = 0; i < n_1; i++$ **do**
11: $\quad\quad (u^{(0)}, u^{(1)}) \leftarrow (u^{(0)}, u^{(1)}) + (a^{(i)}, b^{(i)}) M^{(n_1 j+i)}_{\mathsf{diag}}$
12: $\quad$ **end for**
13: $\quad u^{(1)} \leftarrow \lceil P^{-1} u^{(1)} \rfloor$
14: $\quad \mathbf{d} \leftarrow [[u^{(1)}]_{q_{\gamma_0\leq i<\beta}}]_{PQ_i}$
15: $\quad \mathtt{ct}[0] \leftarrow \mathtt{ct}[0] + \Phi_{n_1 j}(u^{(0)} + \langle \mathbf{d}, \mathsf{rot}^0_{q_{\gamma_i},\Phi_k}\rangle)$
16: $\quad \mathtt{ct}[1] \leftarrow \mathtt{ct}[1] + \Phi_{n_1 j}(\langle \mathbf{d}, \mathsf{rot}^1_{q_{\gamma_i},\Phi_k}\rangle)$
17: **end for**
18: $\mathtt{ct} \leftarrow (\lceil P^{-1}\mathtt{ct}[0] \rfloor, \lceil P^{-1}\mathtt{ct}[1] \rfloor)$
19: **return** $\mathtt{Rescale}(\mathtt{ct})$

Algorithm 4 Coefficients To Slots

Require: $\mathtt{ct}$, $\mathsf{SF}^{-1} = \prod_{i=0}^{k-1} \mathsf{SF}_i^{-1}$.

1: **for** $i \leftarrow 0$ **to** $k-1$ **do**
2: $\quad \mathtt{ct} \leftarrow \mathtt{MatMul}(\mathtt{ct}, ((\frac{1}{2})^{\frac{1}{k}} \mathsf{SF}_i^{-1}))$
3: $\quad \mathtt{ct} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
4: **end for**
5: $\mathtt{ct}_{\mathsf{conj}} \leftarrow \mathsf{Conjugate}(\mathtt{ct})$
6: **return** $\mathtt{ct} + \mathtt{ct}_{\mathsf{conj}}, -i(\mathtt{ct} - \mathtt{ct}_{\mathsf{conj}})$

Algorithm 5 Slots To Coefficients

Require: $\mathtt{ct}^{(0)}, \mathtt{ct}^{(1)}$, $\mathsf{SF} = \prod_{i=0}^{k-1} \mathsf{SF}_i$.

1: $\mathtt{ct} \leftarrow \mathtt{ct}^{(0)} + i \cdot \mathtt{ct}^{(1)}$
2: **for** $i \leftarrow 0$ **to** $k-1$ **do**
3: $\quad \mathtt{ct} \leftarrow \mathtt{MatMul}(\mathtt{ct}, \mathsf{SF}_i)$
4: $\quad \mathtt{ct} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
5: **end for**
6: **return** $\mathtt{ct}$

Algorithm 6 `EvalMod1`

Require: $\{\mathtt{ct}, \ell + 1, \Delta\}$ a ciphertext, $p(t)$ a degree d Chebyshev interpolant of $\left(\frac{q_0}{2^{\lceil \log(q_0) \rfloor}} \frac{1}{2\pi}\right)^{2^{-r}} \cos\left(2\pi \frac{(x-0.25)}{2^r}\right)$, K the range of the interpolation.
Ensure: The evaluation $\{\mathtt{ct} = q_0 \cdot p(\mathtt{ct}/q_0), \ell - \lceil \log(d) \rceil - r, \Delta\}$
1: $\{\mathtt{ct}, \ell + 1, \Delta \cdot q_{\ell+1}\} \leftarrow \mathtt{MultConst}(\mathtt{ct}, 2^{\lceil \log(q_0) \rfloor}/q_0)$
2: $\{\mathtt{ct}, \ell, \Delta\} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
3: $\mathtt{ct} \leftarrow \mathtt{AddConst}(\mathtt{ct}, -0.25/(2^r K))$
4: $\Delta_0 \leftarrow \Delta$
5: **for** $i \leftarrow 0$ **to** $r - 1$ **do**
6: $\quad \Delta_{i+1} \leftarrow \sqrt{\Delta_i \cdot q_{\ell - \log(d-1) - r + i}}$
7: **end for**
8: $\{\mathtt{ct}, \ell - \lceil \log(d) \rceil, \Delta_r\} \leftarrow \mathtt{EvaluatePolynomial}(\{\mathtt{ct}, \ell, \Delta\}, p(t), \Delta_r)$
9: $\delta \leftarrow (1/2\pi)^{2^{-r}}$
10: **for** $i \leftarrow 0$ **to** $r - 1$ **do**
11: $\quad \delta \leftarrow \delta^2$
12: $\quad \{\mathtt{ct}, \ell - \log(d-1) - i, \Delta_{r-i}^2 = \Delta_{r-i-1} \cdot q_{\ell - \log(d-1) - i}\} \leftarrow \mathtt{Relinearize}(\mathtt{Tensor}(\mathtt{ct}, \mathtt{ct}))$
13: $\quad \mathtt{ct} \leftarrow \mathtt{MultConst}(\mathtt{ct}, 2)$
14: $\quad \mathtt{ct} \leftarrow \mathtt{AddConst}(\mathtt{ct}, -\delta)$
15: $\quad \{\mathtt{ct}, \ell - \log(d-1) - (i+1), \Delta_{r-i-1}\} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
16: **end for**
17: **return** $\{\mathtt{ct}, \ell - \log(d-1) - r, \Delta_0 = \Delta\}$

Algorithm 7 `EvaluatePolynomial`

Require: A target scale Δ', a ciphertext $\{\mathtt{ct}, \ell, \Delta\}$ and a polynomial $p(t) = \sum_{i=0}^{d} c_i T_i(t)$.
Ensure: $\{p(\mathtt{ct}), \ell - \lceil \log(d) \rceil, \Delta'\}$.
1: $m \leftarrow \lceil \log(d) \rceil$
2: $l \leftarrow \lfloor m/2 \rfloor$
3: $T(\mathtt{ct}) \leftarrow \{\{1, \mathtt{ct}, T_2(\mathtt{ct}), \ldots, T_{2^{l-1}}(\mathtt{ct})\}, \{T_{2^l}(\mathtt{ct}), T_{2^{l+1}}(\mathtt{ct}), \ldots, T_{2^{m-1}}(\mathtt{ct})\}\}$
4: Express $p(t)$ as $\sum_{i=0}^{2^{m-l}-1} p_i(t) \cdot T_{i2^l}(t)$
5: $\boldsymbol{\Delta}' \leftarrow \mathtt{GetScalingFactors}(p(t), T(\mathtt{ct}), \Delta')$ // Algorithm 8
6: $b \leftarrow \{\emptyset\}$
7: **for** $i \leftarrow 0$ **to** $2^{m-l} - 1$ **do**
8: $\quad b_i \leftarrow \mathtt{BabyStep}(p_i(t), T, \boldsymbol{\Delta}'_i)$ // Algorithm 9
9: **end for**
10: **for** $i \leftarrow 0$ **to** $m - l - 1$ **do**
11: $\quad$ **for** $j \leftarrow 0$ **to** $(2^{m-l})/(2^{i+1}) - 1$ **do**
12: $\quad\quad b_j \leftarrow \mathtt{GiantStep}(b_{2j}, b_{2j+1}, T_{2^{l+i}})$ // Algorithm 10
13: $\quad$ **end for**
14: **end for**
15: **return** $\mathtt{Rescale}(\mathtt{Relinearize}(b_0))$

Algorithm 8 `GetScalingFactors`

Require: A polynomial $p(t) = \sum_{i=0}^{2^{m-1}-1} p_i(t) T_{i2^l}(t)$, a pre-computed power-basis $\mathtt{T} = \{\{T_1(\mathtt{ct}), T_2(\mathtt{ct}), \ldots, T_{2^{l-1}}(\mathtt{ct}), T_{2^l}(\mathtt{ct}), T_{2^{l+1}}(\mathtt{ct}), \ldots, T_{2^{m-1}}(\mathtt{ct})\}\}$ and a target scale Δ'.
Ensure: $\boldsymbol{\Delta} = \{\Delta'_0, \Delta'_1, \ldots, \Delta'_{2^{m-1}-1}\}$ the vector of target scaling factor for the `BabyStep` such that each call of `EvaluateMonomial` in `GiantStep` has valid inputs and the final scale of `EvaluatePolynomial` is Δ'.

Algorithm 9 `BabyStep`

Require: A polynomial $p(t) = \sum_{i=0}^{d} c_i T_i(t)$, a power basis $T = \{T_1(\mathtt{ct}), \mathtt{ct}, T_2(\mathtt{ct}), \ldots, T_d(\mathtt{ct})\}$ and a polynomial and target scale Δ.
Ensure: $\mathtt{ct}' = \{p(\mathtt{ct}), \ell_{T_d(\mathtt{ct})}, \Delta \cdot q_{\ell_{T_d(\mathtt{ct})}}\}$.

1: $\mathtt{ct}' \leftarrow \left\lceil c_0 \cdot \Delta \cdot q_{\ell_{T_d}} \right\rfloor$ // Set to be at level ℓ
2: **for** $i \leftarrow 1$ **to** $d-1$ **do**
3: $\quad \mathtt{ct}'' \leftarrow \mathtt{MultConst}\left(\frac{c_i \Delta}{\Delta_{T_i(\mathtt{ct})}}, T_i(\mathtt{ct})\right)$ // `MultConst` scales c_i by $q_{\ell_{T_d}}$
4: $\quad \mathtt{ct}' \leftarrow \mathtt{Add}(\mathtt{ct}', \mathtt{ct}'')$ // $\mathtt{ct}'$ and a both have scaling factor $\Delta \cdot q_{\ell_{T_d}}$
5: **return** $\mathtt{ct}'$

Algorithm 10 `GiantStep`

Require: Ciphertexts $\{\mathtt{ct}_0, \ell - 1, \Delta_0 = \Delta_1\Delta_2\}, \{\mathtt{ct}_1, \ell, \Delta_1 q_\ell\}, \{\mathtt{ct}_2, \ell - 1, \Delta_2\}$.
Ensure: $\{\mathtt{ct}' = \mathtt{ct}_0 + \mathtt{ct}_1\mathtt{ct}_2, \ell - 1, \ell_2), \Delta_0\}$

1: $\mathtt{ct}_1 \leftarrow \mathtt{Relinearize}(\mathtt{ct}_1)$ //$\Delta_1 q_{\ell_1}$
2: $\mathtt{ct}_1 \leftarrow \mathtt{Rescale}(\mathtt{ct}_1)$ //$\Delta_1 q_{\ell_1} \rightarrow \Delta_1$
3: $\mathtt{ct}_1 \leftarrow \mathtt{Tensor}(\mathtt{ct}_1, \mathtt{ct}_2)$ //$\Delta_1 \rightarrow \Delta_1\Delta_2$
4: $\mathtt{ct}_1 \leftarrow \mathtt{Add}(\mathtt{ct}_1, \mathtt{ct}_0)$ //$\Delta_0 = \Delta_1\Delta_2$
5: **return** $\mathtt{ct}_1$