Practical q-IND-CPA-D-Secure Approximate Homomorphic Encryption

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

[1] Independent
[2] Norwegian University of Science and Technology (NTNU), Norway
[3] Hasso Plattner Institut (HPI), Universität Potsdam, Germany
[4] Tune Insight SA

jeanphilippe.bossuat@gmail.com, anamaria.costache@ntnu.no, christian.mouchet@hpi.de, lea.nurnberger@ntnu.no, juan@tuneinsight.com

Abstract. At Eurocrypt 2021, Li and Micciancio demonstrated that the IND-CPA notion of security is not sufficient to cover the passive security of approximate homomorphic encryption schemes, by outlining a key recovery attack against the CKKS scheme (Cheon, Kim, Kim, Seong, Asiacrypt 2017). They proposed the notion of q-IND-CPA-D security, which allows an adversary to make q calls to a restricted decryption oracle. Li and Micciancio left achieving q-IND-CPA-D security as an open problem, but proposed two approaches: noise flooding and an exact version of CKKS. The first approach was addressed by Li, Micciancio, Schultz and Sorrell (Crypto 2022), but leads to substantial efficiency loss. In this work, we look at the second approach. We define (δ, r)-exact CKKS, a version of CKKS that returns exact results on all except the least r significant bits with (high) probability δ, based on bounds on the noise. We prove that the advantage of a q-IND-CPA-D attacker against (δ, r)-exact CKKS is determined by the failure probability of those bounds. We conduct a tight average-case and implementation-specific noise analysis of all elementary operations in CKKS, as implemented in the Lattigo library, including the bootstrapping operation. We propose bounds that have small enough failure probability for the advantage of a q-IND-CPA-D attacker against (δ, r)-exact CKKS to become smaller than 2^{-128}, while the parameter sets needed remain practical. We furthermore present an estimator tool that combines the bounds on basic operations and returns tight noise estimates, even for large circuits. We validate our bounds by showcasing experimental results on different iterative algorithms, homomorphic encoding, decoding and bootstrapping.

1 Introduction

Fully Homomorphic Encryption (FHE) allows to perform meaningful computations on encrypted data. Following Gentry's initial construction from standard lattices assumptions, current constructions [7, 23, 24, 22, 15, 14] are based on variants of the Learning-with-Error problem (LWE) [36], and 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.

One of the most efficient schemes to date for computing on approximate real and complex numbers was introduced by Cheon et al. [14] 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. For a ciphertext $\mathtt{ct}$ that encrypts a message m, the CKKS decryption outputs $\mathtt{Dec}(\mathtt{ct}) = m + e$, without attempting to filter out the noise e. On the other hand, *exact* FHE schemes [7, 23, 24, 22, 15] apply more advanced encoding strategies, which enable exact decoding of m with high probability, but can result in a performance loss compared to CKKS.

It was shown by Li and Micciancio at Eurocrypt 2021 [33] that knowledge of the error term in CKKS can be used to mount a passive key recovery attack on CKKS, even if CKKS is provably IND-CPA secure. This attack leverages the approximate nature of CKKS, and assumes the attacker is able to obtain decryptions of encrypted plaintexts that are known to them. We describe the attack based on the secret key version of CKKS since this illustrates the key idea of the attack more clearly, and note that the public-key version can be found in [33]. Let $\mathtt{ct} = (-as + e, a) = (\mathtt{ct}[0], \mathtt{ct}[1])$ be a secret key CKKS ciphertext encrypting 0. Assume the attacker has knowledge of $\mathtt{ct}[0]$ and $\mathtt{Dec}(\mathtt{ct})$. Then it calculates $\mathtt{ct}[0] - \mathtt{Dec}(\mathtt{ct}) = \mathtt{ct}[0] - \langle \mathtt{ct}, \mathtt{sk} \rangle = as + e - (as + e - as) = as$, which is a linear equation in the secret key that can easily be solved by multiplying with a^{-1}, which is public. This inverse exists with high probability. If it does not, the adversary can simply make new decryption queries until it recovers a sample where the inverse does exist. Given the fact that CKKS *is* IND-CPA secure, the above attack shows that this notion does not capture the passive security of CKKS (or indeed of any approximate scheme) accurately. Li and Micciancio therefore proposed the notion of q-IND-CPA-D security, that captures the scenario of their attack. The q-IND-CPA-D game extends the capabilities of an adversary, by also allowing it to call an evaluation oracle, and q times a restricted decryption oracle. The decryption oracle can only be called on ciphertexts that have been returned by the encryption or evaluation oracle upon querying for a message pair (m_0, m_1), and only if $m_0 = m_1$, so as to not trivially leak information about the challenge bit b. Provably achieving q-IND-CPA-D security for CKKS was left as an open problem in [33], but two approaches were proposed: noise flooding and converting CKKS into an exact scheme. The first approach has been addressed by [34], where the authors show that noise exponential in the security parameter needs to be added for the noise flooding to achieve provable security, which has a significant negative impact on the efficiency of the scheme.

In this work, we explore the second approach, first formalised by [17]. We show how to obtain a provably q-IND-CPA-D secure scheme, which returns all except for the least r significant bits of the message, with probability δ, by using an average-case noise analysis. This scheme, which we call (δ, r)-exact CKKS, uses noise bounds to round off the r message bits that have been affected by the noise. We show that the advantage of an attacker against the q-IND-CPA-D security of (δ, r)-exact CKKS is determined by the failure probability $1-\delta$ of the bounds. We note that our results in particular validate two recent works [8, 10]. In those works, the authors run a q-IND-CPA-D attack on *exact* schemes precisely by exploiting the high decryption failure probability.

1.1 Our Contributions

We make the following contributions:

- **Provably Secure q-IND-CPA-D secure CKKS.** We construct a (δ, r)-exact CKKS scheme that is provably q-IND-CPA-D secure. We achieve this by rounding off the noise after the CKKS decryption, based on probabilistic noise bounds.
- **Composable Average-Case Noise Bounds.** In order to achieve the above, 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. Thanks to these extensions, we can derive bounds on complex circuits such as iterative algorithms or bootstrapping, which was left unaddressed by previous works.
- **Open-Source Average-Case Noise Estimator.** We provide a proof of concept open-source estimator[5] that calculates these noise bounds for any circuit. This contribution can be of independent interest, as it provides a crucial development tool to assess how FHE circuits behave, or fine tune parameters in order to achieve a target precision for a given application. Our estimator targets (and is implemented with) the Lattigo library [1].
- **Empirical Validation of the Average-Case Noise Bounds.** We perform several experiments to show that the bounds computed with our estimator are tight. For example, we show that the estimated noise standard deviation for the bootstrapping circuit differs by only 0.01 bits from the experimentally obtained one. As a by-product, we therefore provide the first tight noise analysis for CKKS bootstrapping.
- **Practical Instantiatiation of q-IND-CPA-D CKKS.** We use the obtained estimates to demonstrate that there exist practical parameter sets that are both IND-CPA and q-IND-CPA-D secure. In fact, we show that we only lose few bits of precision when instantiating CKKS as a (δ, r)-exact q-IND-CPA-D secure scheme, instead of as an IND-CPA secure scheme, for the same parameter sets.

[5] The code will be made available shortly.

1.2 Related Work

Costache et al. [18] provided the first average-case noise analysis for CKKS. They provided noise estimates for encoding, encryption, addition, multiplication, relinearization and rescaling for both CKKS and RNS-CKKS. They also formalized the approach of achieving exact CKKS through rounding off the noise bits. We build our construction on top of this formalization. In addition to the noise estimates given in [18], we also perform noise analysis for addition and multiplication by plaintexts and constants, key switching in general, as well as rotations and conjugations. We use a slightly different approach, by tracking the component-wise noise (Definition 2), instead of the precision loss directly. We furthermore demonstrate that our noise estimates can be combined into estimates for deeper circuits such as bootstrapping and give experimental validation for this claim.

In the work of [33], the authors proved that IND-CPA and q-IND-CPA-D security are equivalent for exact schemes. It was therefore believed that q-IND-CPA-D attacks are not an issue for schemes such as BGV, BFV, THFE or FHEW. However, two recent works [8, 10] have called this into question. Indeed, they point out that the initial proof of equivalence between IND-CPA and q-IND-CPA-D for exact schemes does not take decryption failures into account. Hence, they are vulnerable to q-IND-CPA-D attacks, where the success probability of the attack depends on the probability of the noise bounds failing. By amplifying this probability through correlated inputs, the authors (succesfully) mount a key recovery attack on the BGV scheme. They also note that this attack applies to most implementations of exact schemes. We confirm these results by providing a formal security proof for CKKS that arrives at the same conclusions.

Recently, [25] showed that using average-case noise analysis for determining the noise flooding noise can lead to an attack, if the noise estimates underestimate the actual noise. However, their attack exploits the fact that the noise analysis assumes independent inputs (hence can be *tricked* into outputting smaller values by using correlated inputs). However our estimator correctly models operations on correlated inputs, which prevents the scenario of the attack [25].

Lastly, [2] introduced the concept of *application-aware FHE*, a model which aims to reflect how FHE schemes are used in real world applications. They show that in such a model, where FHE schemes are only used for the class of circuits they where originally parameterized for, IND-CPA and IND-CPA-D security are equivalent and all previous attacks, such as the ones of [8, 10, 25], are invalid since they would require to evaluate a class of circuit that the scheme was not originally parameterized for. Instantiating their setting starts with an offline procedure which requires to produce a tight noise estimate for the class of circuits that can be run by the evaluator.

Although our work takes a different approach to secure approximate homomorphic encryption, it still nicely complements the one of [2]. This is since among our contributions, we provide tight noise estimates for the CKKS scheme, and our (δ, r)-exact CKKS scheme can be used within their framework to further reduce the probability of information leakage.

1.3 Our techniques

In this work, we construct (δ, r)-exact CKKS, a CKKS variant that is secure in the q-IND-CPA-D model. To do so, we need three building blocks: tight noise bounds for arbitrary circuits in CKKS, a definition of (δ, r)-exact CKKS, and a proof that such a CKKS variant is q-IND-CPA-D-secure. We now provide an overview of the techniques for each building block, and informally state our results for ease of exposition.

Average-case Noise Analysis (Section 3) We derive tight noise estimates for all the elementary operations. Costache et al. have used the notion of precision loss to capture what "noise" is in CKKS [18]. Precision loss describes the difference between the decryption of a ciphertext and the plaintext the ciphertext should encrypt.

We propose to track the noise in each ciphertext component instead (Definition 2), and denote the vector of the component-wise noises of $\mathtt{ct}$ as $n(\mathtt{ct}) = (n(\mathtt{ct})[0], \ldots, n(\mathtt{ct})[k])$. We choose this notion over the notion of precision loss in [18] because tracking the noise in each ciphertext component independently is necessary to obtain precise noise estimates in complex circuits (notably, those that rely on key-switching). We derive (in Section 3) the component-wise noise update rules for each homomorphic operation and, by composition of the latter operations, for arbitrary circuits. We translate the component-wise noise into precision loss only at the end of the evaluation of a circuit as follows

$$[n(\mathtt{ct})[0] + n(\mathtt{ct})[1]s]_Q = [\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q - \mathtt{pt},$$

where $\mathtt{sk} = (1, s)$ is the secret key and Q is the ciphertext modulus. We can obtain the standard deviation of the precision-loss in each coefficient of the plaintext as $\sqrt{\sigma^2_{n(\mathtt{ct})[0]} + N\sigma^2_{n(\mathtt{ct})[1]}\sigma^2_s}$, where N is the polynomial ring degree and $\sigma^2_{n(\mathtt{ct})[k]}$ denotes the noise standard deviation in the k-th component of ciphertext $\mathtt{ct}$. Costache et al. [18] show that the operations in CKKS and the RNS version of CKKS [13] can be well-approximated by a Gaussian distribution. We therefore find that the infinity norm of the precision loss can be bounded as

$$||\mathtt{Decode}([\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q - \mathtt{pt})||_\infty \leq \alpha\sqrt{2n}\sqrt{\sigma^2_{n(\mathtt{ct})[0]} + N\sigma^2_{n(\mathtt{ct})[1]}\sigma^2_s} = B,$$

with probability $\delta = \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$, where $n \leq \frac{N}{2}$ is the number of plaintext slots, $\mathtt{erf}()$ is the error function (see Definition 1), and $\mathtt{Decode}$ is the CKKS decoding procedure defined in Section 2.3. In this work, α is a parameter that allows to tune the tightness and failure probability of the bounds: the smaller α is, the tighter the bounds are, but also the likelier they are to fail.

The authors of [35] have observed that the noise growth differs in different implementations and conjectured that, in order for noise estimates to be tight, they need to be implementation-specific. This conjecture was also made in [18]. The work of [20] confirmed the conjecture for BGV as implemented in HElib. It follows that the implementation of a scheme affects the noise growth substantially. Therefore, during our analysis of the noise in CKKS, we always refer to the algorithms as implemented in Lattigo. We provide the implementation of an estimator of the noise for arbitrary circuits in Lattigo in Section 4, and provide experimental validation for our results. The tightness of these results confirms the conjecture that noise estimates need to be implementation-specific. We note in particular that our estimator allows to also give tight results on more complex circuits such as bootstrapping, Goldschmidt division for 10 iterations, or binary value cleaning as proposed in [21].

(δ, r)-exact CKKS (Section 5) The next building block we need is a definition of (δ, r)-exact CKKS. While such a scheme was conceptualized in the works of [18, 33], a rigorous definition was not provided.

Ideally, we would require the decryption result to be exact except for the least r significant bits of the message, *with probability* 1. The motivation for this is as follows. We cannot know the precise value of the noise term, since this would break the IND-CPA-D security of CKKS immediately. Therefore, if we want to make CKKS exact, we have to remove the r message bits that are polluted by the noise.

However, to achieve this definition in practice, we would need deterministic noise bounds that are guaranteed to hold with probability 1. These bounds exist but are very loose, and using them would substantially reduce the message precision in our r-exact CKKS scheme. We therefore relax this requirement, and define our (δ, r)-exact CKKS scheme as a CKKS scheme that is exact except on the r least significant bits of the message, *with (negligible) probability* δ. This allows us to use average-case bounds, which are much tighter and therefore have a lesser impact on the scheme's efficiency. In order to achieve such a definition of (δ, r)-exact CKKS, we need to introduce a condition of correctability (Definition 8 in Section 5). This condition ensures that rounding the decryption output will always output correct results. By admissible circuits we mean circuits that can be evaluated in CKKS. We obtain the following definition.

Definition 9(δ, r)**-exact CKKS (informal).** *Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}_{\mathsf{exact}}, \mathtt{Eval})$ be a fully homomorphic encryption scheme, where $\mathtt{KeyGen}, \mathtt{Enc}$, and $\mathtt{Eval}$ are the same as the algorithms with the same name in CKKS. The parameters of $\mathcal{E}$ are the same as for CKKS. Let $\mathtt{Dec}'_r$ be a (modified) decryption function. Let m be any message, $b = \log_2(m)$, r some parameter, and $m = \sum_{i=0}^{b-1} m_i 2^i$ be its binary representation. If*

$$\mathtt{Dec}'_r(\mathtt{ct}, \mathtt{sk}) \ = m' = \sum_{i=r}^{b} m_i 2^i$$

for any ciphertext $\mathtt{ct}$ encrypting m, with probability *δ, then we call $\mathcal{E}$ an (δ, r)-exact CKKS scheme.*

q-IND-CPA-D Security of (δ, r)-exact CKKS We finally show (in Section 5) that the following theorem holds, as long as the bound B holds with probability $\delta = \texttt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$, for q the maximum number of calls permitted to the decryption oracle and n the number of plaintext slots.

Theorem 1(Main theorem, informal) *Let $\mathcal{E}$ be the (δ, r)-exact CKKS scheme, and let $\mathcal{A}$ be an adversary against the IND-CPA security of CKKS. Let δ be the probability that the noise bound holds. Let q be the maximum number of the decryption oracle calls allowed. Then we get for the advantage of an adversary $\mathcal{B}$ against the q-IND-CPA-D security of (δ, r)-exact CKKS.*

$$\mathsf{Adv}^{\mathsf{q-IND-CPA-D}}_{(\delta,r)\text{-exact CKKS}}(\mathcal{B}) \leq 1 - \delta^q + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{CKKS}}(\mathcal{A}).$$

We show that there are parameter sets such that the advantage of an adversary $\mathcal{A}$ against the IND-CPA-D security of (δ, r)-exact CKKS is negligible, while retaining practical performance We suggest some concrete parameter sets in Table 2.

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 at most 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 as $\mathbf{v}$. For a polynomial f, we denote its i^{th} coefficient by f_i. 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}. If it is clear that all the coefficients of p are identically distributed, we drop the index i. Let $\mathbf{P} = (p^{(1)}, \ldots, p^{(k)})$ be a vector of random polynomials drawn from $\mathcal{R}_{Q_\ell}$. Then we denote by $\sigma_{\mathbf{P}_i} = (\sigma_{p_i^{(1)}}, \ldots, \sigma_{p_i^{(k)}})$ the vector of the $i-$th coefficient standard deviations of the polynomials in $\mathbf{P}$. Let p be a polynomial. Then $\lceil p \rfloor$ refers to coefficient-wise rounding to the nearest integer; ciphertexts in CKKS are tuples in $\mathcal{R}_Q^{k+1}$, with $k \geq 1$, and we will write them as $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1], \ldots, \mathtt{ct}[k])$. We denote the secret key by $\mathtt{sk} = (1, s, s^2, \ldots, s^k)$. Therefore, we can write the decryption as $[\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q = \left[\sum_{i=0}^{k} \mathtt{ct}[i] \cdot s^i\right]_Q$. Note that for fresh ciphertexts and after most operations $k = 1$. We will therefore in this work assume ciphertexts of length 2 unless explicitly stated otherwise.

2.3 Messages and Plaintexts

For reasons of space, we delay the presentation of the CKKS scheme to Appendix A (Figures 4 and 5). However, we describe the encoding and decoding between the message space $\mathbb{C}^n$ and the plaintext space $\mathcal{R}_Q$.

The encoding of a message $\mathbf{m} \in \mathbb{C}^n$ is performed 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*.

Through scaling and rounding, the element $\tau^{-1}(\mathbf{m})$ is discretized onto an element $\mathtt{pt}$ in the ring $\mathcal{R}_{Q_\ell}$; τ^{-1} is computed as follows. Let $\mathbf{m} \in \mathbb{C}^n$ and compute $\mathbf{m}' = \mathtt{DFT}_n^{-1}(\mathbf{m})$ the inverse Discrete Fourier Transform (DFT) of $\mathbf{m}$. 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 $\mathtt{DFT}_n^{-1}(\mathbf{m})$ and the second n coefficients the imaginary part of $\mathtt{DFT}_n^{-1}(\mathbf{m})$. To transform $\tau^{-1}(\mathbf{m})$ into an element in $\mathcal{R}_{Q_\ell}$, calculate $\mathtt{pt} = \lceil \Delta \cdot \tau^{-1}(\mathbf{m}) \rfloor$ for a scaling factor Δ, and apply the change of variable $Y \to X^{N/2n}$. In what follows, we will call elements in the ring $\mathcal{R}_{Q_\ell}$ plaintexts, and elements in $\mathbb{C}^n$ messages. We will denote them by $\mathtt{pt} = \lceil \Delta \cdot \tau^{-1}(\mathbf{m}) \rfloor$ and $\mathbf{m}$ respectively.

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 $\mathbb{R}[Y + Y^{-1}]/(Y^{2n}+1) \subset \mathbb{R}[Y]/(Y^{2n}+1) \subseteq \mathbb{R}[X]/(X^N+1)$.

2.4 Products of Random Polynomials and Error Function

We give two results on the coefficient standard deviation of products of random polynomials that we will need later on. From here on, we use the term “random polynomial” to refer to a polynomial whose 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 [20].

Lemma 1 (Products of Random Polynomials [20]). *Let $f, g \in \mathcal{R}, f \neq g$ be two polynomials of degree N, whose coefficients are drawn identically and independently from two distributions $\mathcal{D}_f$ and $\mathcal{D}_g$, with variance σ_f^2 and σ_g^2 respectively and mean μ_f and μ_g respectively.*

$$f_i \xleftarrow{i.i.d} \mathcal{D}_f, \quad g_i \xleftarrow{i.i.d} \mathcal{D}_g,$$

$i \in \{0, \ldots, N-1\}$, where μ_j is the mean and σ_j^2 is the variance of $\mathcal{D}_j$ respectively. Let $\mathbb{E}(\mathcal{D}_j)$ denote the expectation of $\mathcal{D}_j$, $j \in \{f, g\}$. Then the variance of the distribution of the coefficients of $f \cdot g$ is:

$$\sigma^2_{(fg)_i} = N(\mathbb{E}(\mathcal{D}_f)^2\sigma_g^2 + \mathbb{E}(\mathcal{D}_g)^2\sigma_f^2 + \sigma_g^2\sigma_f^2).$$

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, 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 $(f^2)_i, i \in \{0, \ldots, N-1\}$*

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

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

Proof. Deferred to Appendix B □

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*

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

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 $1 - \texttt{erf}\left(\frac{14}{\sqrt{2}}\right) \approx 2^{-145.5}$.

2.5 Noise Definitions

Noise analysis in CKKS comes with two extra challenges: the 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 ct encrypting

m, the decryption does not return m but a noisy term $m+e$ that differs from the original message. It is this difference that we consider as noise. The authors of [18] calculate this as $[\langle \texttt{ct}, \texttt{sk}\rangle]_Q - \texttt{pt}$ before decoding and as $\texttt{Decode}([\langle \texttt{ct}, \texttt{sk}\rangle]_Q) - m$ after decoding, for $\texttt{ct}$ encrypting $\texttt{pt}$, and $\texttt{pt}$ the encoding of m. The work of [18] tracked the precision loss through a computation in CKKS by tracking the variance of $[\langle \texttt{ct}, \texttt{sk}\rangle]_Q - \texttt{pt}$. We will follow a different approach and track the coefficient standard deviation of the noise in each component, that then can be combined in a coefficient standard deviation of the precision loss.

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

In other words, the component-wise noise captures all the terms introduced during encryption and homomorphic evaluation which do not get cancelled out during decryption. Note that this includes the error introduced by encoding, where relevant. The precision loss $\langle \texttt{ct}, \texttt{sk}\rangle - \texttt{pt}$ can be obtained from the component-wise noise as $\sum n(\texttt{ct})[i] \cdot s^i$. We use the component-wise definition because the noise coming from the key-switching operation does not depends on $n(\texttt{ct})[0]$ and thus requires to be able to distinguish between the different components.

2.6 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 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 [16]). Due to the bounding after each step in the circuit, worst-case bounds are very loose for large circuits, since the approximation errors accumulate, as observed in [19].

The average-case analysis has 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 bound at the very end. The resulting bounds are tighter, but can have a larger probability failure than worst-case bounds. The authors of [18] showed that the noise after each operation in CKKS can be well approximated by a Gaussian distribution in each component, assuming independent inputs. Why theoretically assuming independency is a stretch, this assumption can be experimentally validated (see for example [20]). Therefore, the infinity norm of a complex vector of length n can be shown to lie with probability $\delta = \texttt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$ in an interval of α standard deviations around 0 (we consider a complex vector as a real vector of twice the size). Since the authors take the approach of calculating the standard deviation of the whole circuit and only bound the error at the very end, they do not incur a bounding error at every step, but only once. This leads to much more precise error bounds and the ability to tightly control the failure probability of the bounds. In this work, we use the latter 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 n. Let $\mathtt{sk} = (1, s, s^2, \dots, s^k)$ be the secret key, and σ_{s^k} its standard deviation. Then we have*

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

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

2.7 Security Notions and Definitions

In this work we will need the following two security definitions. IND-CPA-D is a security notion that has been proposed by Li and Micciancio [33] to capture the scenario of their passive attack on CKKS.

Definition 3 (IND-CPA Security). *Let $\mathcal{E} = (\mathtt{KeyGen}_\mathcal{E}, \mathtt{Enc}_\mathcal{E}, \mathtt{Dec}_\mathcal{E}, \mathtt{Eval}_\mathcal{E})$ be a fully homomorphic encryption scheme. We define an experiment $\mathsf{Expr}^{\mathsf{IND-CPA}}[\mathcal{A}]$ an efficient adversary $\mathcal{A}$.*

```
$\underline{\mathsf{Expr}^{\mathsf{IND-CPA}}[\mathcal{A}](\lambda):}$
$b \xleftarrow{\$} \{0,1\}$
$(\mathtt{sk}, \mathtt{pk}, \mathtt{evk}) \leftarrow \mathtt{KeyGen}(\lambda)$
$S \leftarrow \emptyset$
$i \leftarrow 0$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathtt{pk}}}(\lambda)$
return $b' = b$,
```

where $\mathsf{Enc}_{\mathtt{pk}}$ is the encryption oracle defined in Figure 1. We say that $\mathcal{E}$ is IND-CPA secure if

$$\mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathcal{E}}(\mathcal{A}) = \left|\mathsf{Pr}\left(\mathsf{Expr}^{\mathsf{IND-CPA}}[\mathcal{A}](\lambda) \Rightarrow 1 \middle| \mathsf{Expr}^{\mathsf{IND-CPA}}[\mathcal{A}](\lambda) \not\Rightarrow \perp\right) - \frac{1}{2}\right|$$

is negligible in λ.

```
$\underline{\mathsf{Enc}^b_{\mathsf{pk}}(m_0, m_1):}$
$\mathtt{ct} \leftarrow \mathtt{Enc}_{\mathcal{E}}(m_b, \mathtt{pk})$
return $\mathtt{ct}$
```

Fig. 1: Oracle for the IND-CPA game.

Definition 4 (q-IND-CPA-D Security [33]). *Let $\mathcal{E} = (\mathit{KeyGen}_{\mathcal{E}}, \mathit{Enc}_{\mathcal{E}}, \mathit{Dec}_{\mathcal{E}}, \mathit{Eval}_{\mathcal{E}})$ be a fully homomorphic encryption scheme. We define an experiment $\mathsf{Expr}_b^{\mathsf{q-IND-CPA-D}}$, parametrised by an efficient adversary $\mathcal{A}$, where the adversary can make at most q calls to the decryption oracle.*

```
$\underline{\mathsf{Expr}^{\mathsf{q-IND-CPA-D}}[\mathcal{A}](\lambda):}$
$b \xleftarrow{\$} \{0,1\}$                    $(sk, pk, evk) \leftarrow KeyGen(\lambda)$
$S \leftarrow \emptyset$
$i \leftarrow 0$
$\tilde{q} \leftarrow 0$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk}, \mathsf{Eval}_{evk}}(\lambda)$
return $b' = b$,
```

where $\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk}$, and Eval_{evk} are the encryption, decryption, and evaluation oracle respectively as defined in Figure 2. Then $\mathcal{E}$ is said to be q-IND-CPA-D secure if

$$\mathsf{Adv}_{\mathcal{E}}^{\mathsf{q-IND-CPA-D}}(\mathcal{A}) = \left| \mathsf{Pr}\left(\mathsf{Expr}^{\mathsf{q-IND-CPA-D}}[\mathcal{A}](\lambda) \Rightarrow 1 \middle| \mathsf{Expr}^{\mathsf{q-IND-CPA-D}}[\mathcal{A}](\lambda) \not\Rightarrow \perp \right) - \frac{1}{2} \right|$$

is negligible in λ.

For the proof in Section 5 we will need the following definitions and lemmas.

Definition 5 (Advantage of Games). *Let $\mathcal{G}_0$ and $\mathcal{G}_1$ be two games, and $\mathcal{A}$ an adversary. We define their advantage as*

$$\mathsf{Adv}(\mathcal{G}_0^{\mathcal{A}}, \mathcal{G}_1^{\mathcal{A}}) := |\mathsf{Pr}(\mathcal{G}_0^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_0^{\mathcal{A}} \not\Rightarrow \perp) - \mathsf{Pr}(\mathcal{G}_1^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_1^{\mathcal{A}} \not\Rightarrow \perp)|.$$

Definition 6 (Identical-until-bad-games [3]). *Two games $\mathcal{G}_0$ and $\mathcal{G}_1$ are called identical-until-bad-games if they are syntactically equivalent until a flag BAD is set to true.*

Lemma 5 (Difference Lemma [3]). *Let $\mathcal{A}$ be an adversary and let $\mathcal{G}_0$, $\mathcal{G}_1$ be identical-until-bad-games. Then*

$$\mathsf{Adv}(\mathcal{G}_0^{\mathcal{A}}, \mathcal{G}_1^{\mathcal{A}}) \leq \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_0}) = \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_1}).$$

```
$\mathsf{Enc}^b_{\mathsf{pk}}(m_0, m_1)$ :
$\mathtt{ct} \leftarrow \mathtt{Enc}_{\mathcal{E}}(m_b, \mathtt{pk})$
$S \leftarrow (m_0, m_1, \mathtt{ct})$
$i \leftarrow i + 1$
return $\mathtt{ct}$

$\mathsf{Eval}^b_{\mathsf{evk}}(g, J = (j_1, \dots, j_\ell))$ :
$\mathtt{ct} \leftarrow \mathtt{Eval}_{\mathcal{E}}(g, S[j_1].\mathtt{ct}, \dots, S[j_\ell].\mathtt{ct}, \mathtt{evk})$
$g_{m_0} \leftarrow g(S[j_1].m_0, \dots, S[j_\ell].m_0)$
$g_{m_1} \leftarrow g(S[j_1].m_1, \dots, S[j_\ell].m_1)$
$S[i] \leftarrow (g_{m_0}, g_{m_1}, \mathtt{ct})$
$i \leftarrow i + 1$
return $\mathtt{ct}$

$\mathsf{Dec}^b_{\mathsf{sk}}(i)$ :
If $S[i].m_0 = S[i].m_1$ and $\tilde{q} \leq q$ :
    $\tilde{q} \leftarrow \tilde{q} + 1$
    return $\mathtt{Dec}_{\mathcal{E}}(S[i].\mathtt{ct}, \mathtt{sk})$
Else:
    return $\perp$
```

Fig. 2: Oracles for the q-IND-CPA-D game.

3 Average-case Noise Estimates for RNS-CKKS

In the following, we calculate the coefficient standard deviation of the component-wise noise for the basic operations in RNS-CKKS. Lemmas 18 - 26 show how to combine the standard deviations of the basic operations into standard deviations of the bootstrapping circuit as an example for a complex circuit. Due to space reasons, these Lemmas can be found in Appendix C. These noise estimates will then be used in the estimator to allow for the automated calculation of the precision loss during the evaluation of a circuit, before this evaluation has started.

We will use the following notation in this section. 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 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 encoding noise standard deviation $\sigma_{n(\mathtt{pt})_i}$. Whenever an operation is performed with a plaintext 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 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 $\mathtt{Decode}(\mathtt{Encode}(\mathbf{m})) \approx \mathbf{m}$. The difference between $\mathtt{Decode}(\mathtt{Encode}(\mathbf{m}))$ 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 $\frac{1}{12}$ [18]. We denote the encoding error as $n(\mathtt{pt})$, that is, we have $\mathtt{pt} + n(\mathtt{pt}) = \left\lceil \Delta\tau^{-1}(\mathbf{m}) \right\rfloor$.

3.2 Noise Estimates for Encryption and Rescaling

For the encryption, the encoding noise is still separated from the ciphertext noise coming from the encryption randomnesses. We therefore will here still track it separately. After computations have been performed on fresh ciphertexts, the encoding and the ciphertext noise get intertwined and both will contribute to the component-wise noise $n(\mathtt{ct})$. For example, during a multiplication, the ciphertext noise gets multiplied by both the plaintext and the plaintext noise. As such, we will track this component-wise noise for all remaining operations instead of the single-valued *critical-quantity*. As we will later see, this is notably required to obtain accurate estimates for the key-switching noise. As described above, the plaintext will impact the noise. It therefore becomes necessary to discuss how to model the plaintext. There are two scenarios for calculating noise estimates: to determine the noise and the parameters for a general circuit, or to determine them for a concrete calculation. In the latter, the messages and therefore plaintexts on which the calculations will be run can be assumed to be known. In the former, they cannot. The first case makes it necessary to assume that the messages, and therefore the plaintexts, are drawn from a bounded distribution with standard deviation $\sigma_{\mathtt{pt}}$, and to consider them a random variable themselves. 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.

In the second case, the plaintexts can be considered deterministic and not a random variable, and therefore do not have a standard deviation. In what follows, we will give the noise estimates for the more general case, where the plaintext is drawn from a distribution and is therefore modelled by its standard deviation. The noise estimates can be adapted to the latter case as follows: Let X, Y be two random vectors with $2n$ coefficients that are identically and independently distributed, and $\mathtt{pt}^{(0)}, \mathtt{pt}^{(1)} \in \mathcal{R}_{Q_\ell}$ two plaintexts.

If $\mathtt{pt}^{(0)}, \mathtt{pt}^{(1)}$ are two random variables independent of X, Y, then the coefficient variance of $\mathtt{pt}^{(0)}X + \mathtt{pt}^{(1)}Y$ is given by Lemma 1 as $N\sigma^2_{\mathtt{pt}_i^{(0)}}\sigma^2_{X_i} + N\sigma^2_{\mathtt{pt}_i^{(1)}}\sigma^2_{Y_i}$. If $\mathtt{pt}^{(0)}, \mathtt{pt}^{(1)}$ are not random variables, then we obtain $(\mathtt{pt}_i^{(0)})^2\sigma^2_X + (\mathtt{pt}_i^{(1)})^2\sigma^2_Y$, where $(\mathtt{pt}_i^{(j)}), \in j\{0, 1\}$ is the $i-$th coefficient of the plaintext polynomial. Thus, the noise estimates for a general circuit can easily be converted into the noise estimates for a specific computation, by switching out $N\sigma^2_{\mathtt{pt}_i^{(j)}}$ for $(\mathtt{pt}_i^{(j)})^2$.

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

Due to space reasons, we cannot give all proofs in this section. We sketch the proofs of which we think they best illustrate our general proof techniques, and fully include those that are sufficiently short. The full proofs of all Lemmas of which a full proof is not stated in this section can be found in Appendix B.

Lemma 6 (Rescaling). *Let $\{\mathtt{ct}, Q_\ell, \Delta\}$ be a ciphertext encrypting a plaintext $\mathtt{pt}$ and let $\mathtt{ct}^{(1)} = \mathtt{DropLevel}(\{\mathtt{ct}, Q_\ell, \Delta\}, k)$ and $\mathtt{ct}^{(2)} = \mathtt{Rescale}(\{\mathtt{ct}, Q_\ell, \Delta\})$, both encrypting $\mathtt{pt}$. Then we have for the standard deviation of the component-wise noise*

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

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

We have dropped the index of the standard deviation, since all the coefficients can be assumed to be independently and identically distributed.

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 7 (Fresh Encryption). *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 $\mathtt{Encrypt()}$ 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, we present a sketch of the proof. The full proof can be found in Appendix B.

– 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 standard deviation $\frac{1}{12}$.

- 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 $a \overset{\$}{\leftarrow} \mathcal{R}_{Q_L}$ 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.

□

3.3 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 8 (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*

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

Proof. Deferred to Appendix B □

Lemma 9 (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*

$$\begin{aligned}\sigma_{n(\mathtt{ct}^{(2)})_{i=0,N/2}} &= \left(\sqrt{\sigma^2_{n(\mathtt{ct}^{(0)})_{i=0,N/2}[0]} + \frac{1}{12}}, \sigma_{n(\mathtt{ct}^{(0)})_{i=0,N/2}[1]}\right) \\ \sigma_{n(\mathtt{ct}^{(2)})_{i\neq0,N/2}} &= \left(\sigma_{n(\mathtt{ct}^{(0)})_{i\neq0,N/2}[0]}, \sigma_{n(\mathtt{ct}^{(0)})_{i\neq0,N/2}[1]}\right).\end{aligned}$$

Proof. Deferred to Appendix B. □

Lemma 10 (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*

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

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 quickly: 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. □

Lemma 11 (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 for the coefficient standard deviation of the component-wise noise*

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

Proof. Deferred to Appendix B. □

Lemma 12 (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. Deferred to Appendix B. □

Lemma 13 (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*

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

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

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

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 7. □

3.4 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 [5] and given in Table 4.

Lemma 14 (Key-Switch).

Let $\mathtt{ct}'$ be a ciphertext and $\mathtt{ct} = \mathtt{ct}' + \mathit{KeySwitch}(\mathtt{ct}'[i], s)$ 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 $n(\mathtt{ct})$

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

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

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

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 B.

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 15 (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 14. □

We base our noise estimates for the rotation on an improved version of the rotation algorithm as proposed in Algorithm 4 in [5], 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 D.

Lemma 16 (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 $\Phi_k(m_{\mathtt{ct}})$ 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*

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

Proof. Since Φ is a permutation that induces a sign change at most, the proof can be trivially derived from the proof of Lemma 14. □

Lemma 17 (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*

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

Proof. Since Φ is a permutation that induces a sign change at most, the proof can be trivially derived from the proof of Lemma 14. □

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. Although such an approach works in the average case for basic operations, it breaks down in practice when we start to consider structured plaintexts or start to compose basic operations. Instead, and similarly to how noise estimate 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 [1]. 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 only starts to be 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 validating 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, making it on par with the time necessary to run the computation in the encrypted domain. We however believe that an optimized and clever implementation could significantly improve its efficiency:

– Most operations in the estimator are Hadamard products or additions

- The most time consuming operation under encryption, the key-switching, is replaced in the estimator by a noise sampling
- Only Δ^2 of precision is required, thus arbitrary precision complex arithmetic is unnecessary since in the vast majority of applications Δ ranges from 2^{20} to 2^{50}. Thus replacing the arbitrary precision complex arithmetic by an implementation based on the bivariate polynomial representation proposed by Georgieva et al. [4] with the recently released spqlios-arithmetic back end[6] should provide greatly improved performance.

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 `STG`, 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 $T_{4096}(\texttt{ct})$, 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.

[6] `https://github.com/tfhe/spqlios-arithmetic`

Set	$\log(Q)$	$\log(P)$	h	L	$\log(\varDelta)$	`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

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 $\log(\varDelta)$ 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`).

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.

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 [21]. The total circuit depth is 14. Note that 7 sequential evaluations is 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 of this experiment 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 precision slightly lower) and 27.4 for $h = 192$ [6]. 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 $\mathsf{EvalMod}$ 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 of this experiment can be found in Set VIII of Table 1.

5 An Exact Version of the CKKS Scheme

In this section we prove the following theorem.

Theorem 1 (Advantage against q-IND-CPA-D of (δ, r)-exact CKKS). *Let $\mathcal{E}$ be the (δ, r)-exact CKKS scheme, and let $\mathcal{A}$ be an adversary against the IND-CPA security of CKKS. Let n be the number of plaintext slots, and δ be the probability that the noise bound holds. Let q be the maximum number of calls allowed to the decryption oracle. Then we get for the advantage of an adversary $\mathcal{B}$ against the q-IND-CPA-D security of (δ, r)-exact CKKS.*

$$\mathsf{Adv}^{\mathsf{q-IND-CPA-D}}_{(\delta,r)\text{-}exact\ CKKS}(\mathcal{B}) \leq 1 - \delta^q + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{CKKS}}(\mathcal{A}).$$

We build on top of the results of [18], and prove that the bounds we developed in this work can be used to achieve a provably q-IND-CPA-D secure (δ, r)-exact CKKS CKKS scheme. We give a formal definition of r-exact CKKS and (δ, r)-exact CKKS.

Definition 7 (r-exact CKKS). *Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}_{\mathsf{exact}}, \mathtt{Eval})$ be a fully homomorphic encryption scheme, where $\mathtt{KeyGen}, \mathtt{Enc}$, and $\mathtt{Eval}$ are the same as the algorithms with the same name in CKKS. The parameters of $\mathcal{E}$ are the same as for CKKS. Let $\mathtt{Dec}'_r$ be a decryption function, such that the following holds. Let m be any message, $b = \log_2(m)$, r some parameter, and $m = \sum_{i=0}^{b-1} m_i 2^i$ be its binary representation. If*

$$\mathtt{Dec}'_r(\mathtt{ct}, \mathtt{sk}) \ = m' = \sum_{i=r}^{b} m_i 2^i$$

for any ciphertext $\mathtt{ct}$ *encrypting* m*, then we call* $\mathcal{E}$ *an* r*-exact CKKS scheme.*

We encounter a further problem when defining an exact CKKS scheme: even a small error can potentially change all the bits in the message, and rounding can therefore lead to a different value than the one originally intended. This requires the additional definition of a correctable circuit. We slightly modify the definition in [18].

Definition 8 (Condition for Δ'-Correctability). *Let* $\mathtt{pt} \in \mathbb{Z}[i]^N$ *be a plaintext, and let* Δ' *be some correction factor. Then we say that* $\mathtt{pt}$ *is* Δ'*-correctable, if* $\frac{1}{\Delta'}\mathtt{pt} \in \mathbb{Z}[i]^N$*. We say that a circuit* $g : (\mathbb{Z}[i]^N)^\ell \to \mathbb{Z}[i]^N$ *is* Δ'*-correctable, if* $g(\mathtt{pt}_1, \ldots, \mathtt{pt}_\ell)$ *is* Δ'*-correctable for all choices of inputs* $\mathtt{pt}_i$*.*

The reason for this definition is as follows: Let $\mathtt{pt} + e$ be a decryption result, where $\mathtt{pt}$ is Δ'-correctable. For simplicity, we only consider one plaintext slot. We cannot know the precise value of e, but we can bound its magnitude. By choosing the correction factor as in the definition above, we can guarantee that $\left|\left|\frac{1}{\Delta'}e\right|\right|_\infty \leq \frac{1}{2}$. Since $\mathtt{pt}$ is Δ'-correctable, we know that $\frac{1}{\Delta'}\mathtt{pt} \in \mathbb{Z}[i]$. Therefore, the rounding of $\frac{1}{\Delta'}(\mathtt{pt} + e)$ only incurs a rounding error from the rounding of e, but not from the rounding of $\mathtt{pt}$. We thus know that $\frac{1}{\Delta'}(\mathtt{pt} + e)$ is not further away from $\frac{1}{\Delta'}\mathtt{pt}$ than $\frac{1}{2}$ in absolute value, and can therefore precisely predict the outcome of the rounding. Now, suppose $\mathtt{pt}$ would not be Δ'-correctable, that is $\frac{1}{\Delta'}\mathtt{pt} \notin \mathbb{Z}[i]$. Then the distance between $\frac{1}{\Delta'}(\mathtt{pt} + e)$ and $\left\lceil \frac{1}{\Delta'}\mathtt{pt} \right\rfloor$ can be larger than $\frac{1}{2}$. Therefore, the rounding of $\frac{1}{\Delta'}(\mathtt{pt} + e)$ can lead to a different result than the rounding of $\frac{1}{\Delta'}\mathtt{pt}$, which is not the case if $\mathtt{pt}$ is Δ'-correctable. Since whether this happens or not is dependent on the plaintext, an attacker will always be able to choose a plaintext such that the rounding of $\frac{1}{\Delta'}(\mathtt{pt} + e)$ produces a different result than the rounding of $\frac{1}{\Delta'}\mathtt{pt}$, no matter which rounding function is used. Therefore, if $\mathtt{pt}$ is not Δ'-correctable, an attacker will always be able to distinguish between an r-exact CKKS scheme, and an approximate CKKS scheme that removes the noise through rounding the least significant bits off.

Thus, without this condition we cannot achieve a r-exact CKKS scheme. We will therefore continue by limiting the admissible circuits for r-exact CKKS to the Δ'-correctable circuits.

We define (δ, r)-exact CKKS as follows. We then prove a bound on the advantage of a q-IND-CPA-D attacker against (δ, r)-exact CKKS.

Definition 9 ((δ, r)-exact CKKS). *Let* δ *be probability of a noise bound* $B = \alpha\sqrt{2n}\sqrt{\sigma^2_{n(\mathtt{ct})[0]} + N\sigma^2_{n(\mathtt{ct})[1]}\sigma^2_s}$ *holding, for* $\alpha \in \mathbb{R}$*. Let* $r = \lceil \log(B) \rceil + 1$*, and let* $\Delta' = 2^r$*. Let* $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}_{\delta,r}, \mathtt{Eval})$ *be a fully homomorphic encryption scheme, parameterized by the same parameters as CKKS, and the additional parameters* δ, B, r*, and* Δ'*, where the algorithms* $\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Eval}$ *are the same as in CKKS and* $\mathtt{Dec}_{\delta,r}$ *is defined as follows*

$$\underline{\mathtt{Dec}_{\delta,r}(\mathtt{ct}, \mathtt{sk})} : \textit{return } \Delta' \left\lceil \frac{1}{\Delta'}\mathtt{Decode}([\langle \mathtt{ct}, \mathtt{sk} \rangle]_Q) \right\rfloor,$$

We define the set of admissible circuits of (δ, r)-exact CKKS to be the subset of admissible circuits in CKKS that are Δ'-correctable.

We now prove Theorem 1.

Proof. We define the games $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2$ and $\mathcal{G}_3$ as in Figure 3. In the games, S is the state that is kept in the q-IND-CPA-D game. We do not strictly need it in the IND-CPA game, but it makes the hops between the games more evident. The superscripts over the adversary declare to which oracles the adversary has access in which games. $\tilde{q}$ keeps track of the current number of calls to the decryption oracle. Then, we notice the following.

- Game $\mathcal{G}_0$ is the IND-CPA game for CKKS. We are additionally keeping a state S, but this does not influence the advantage. We therefore have
$$\mathsf{Adv}_{\mathsf{CKKS}}^{\mathsf{IND-CPA}}(\mathcal{B}) = \mathsf{Adv}^{\mathcal{G}_0}(\mathcal{B}).$$
- The decryption oracle Dec' in Game $\mathcal{G}_1$ is perfectly simulatable by any adversary $\mathcal{B}$ against Game $\mathcal{G}_0$, since it does not require any information about the secret key, except for its variance which is assumed to be publicly known. Equally, the evaluation oracle is perfectly simulatable by any adversary $\mathcal{B}$ against game $\mathcal{G}_0$, since the evaluation key is known to the adversary, and the circuit is assumed to be public. Since the encryption oracle is the same as in Game $\mathcal{G}_0$ we have
$$\mathsf{Adv}^{\mathcal{G}_0}(\mathcal{B}) = \mathsf{Adv}^{\mathcal{G}_1}(\mathcal{B}).$$
- Game $\mathcal{G}_3$ is the q-IND-CPA-D game for (δ, r)-exact CKKS. We therefore have
$$\mathsf{Adv}_{(\delta,r)\text{-exact CKKS}}^{q\mathsf{-IND-CPA-D}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_3}(\mathcal{A}).$$

We next take a look at the relation between Games $\mathcal{G}_1$ and $\mathcal{G}_2$. Let m be a message and $m = \sum_{i=0}^{r+\lceil \log(\Delta') \rceil - 1} m_i 2^i$ its binary representation for some Δ'.

Since the set of admissible circuits in (δ, r)-exact CKKS is limited to circuits that return correctable messages, we have $\frac{1}{\Delta'} m \in \mathbb{Z}[i]$. This implies in particular that $m_i = 0$ for $0 \leq i < r$.

Thus, by definition of Δ'-correctability, the decryption oracle in Game $\mathcal{G}_1$ returns the exact same result as $\mathtt{Dec}_r(\mathtt{ct}, \mathtt{sk}, \Delta')$ for a given ciphertext $\mathtt{ct}$ encrypting m. Since $\mathtt{Dec}_r(\cdot, \mathtt{sk})$ returns the same result no matter whether the if-clause in $\mathsf{Dec}_{\mathtt{sk}}$ is triggered or not, Dec' in Game $\mathcal{G}_1$ and $\mathsf{Dec}_{\mathtt{sk}}$ in Game $\mathcal{G}_2$ return the same results. The games are therefore equivalent, and we have
$$\mathsf{Adv}^{\mathcal{G}_1}(\mathcal{B}) = \mathsf{Adv}^{\mathcal{G}_2}(\mathcal{B}).$$

Lastly, we look at the relation between Games $\mathcal{G}_2$ and $\mathcal{G}_3$. The games are identical except for the decryption oracle. Assume that the bounds hold, that is we have $||\mathtt{Decode}([\langle S[i].\mathtt{ct}, \mathtt{sk}\rangle - S[i].m_b]_Q)||_\infty < B$. Then, we have by the Δ'-correctability of m, the message encrypted by $\mathtt{ct}$, for the output of $\mathtt{Dec}_{\delta,r}(\mathtt{ct}, \mathtt{sk})$ the following

```
Game $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$
$b \leftarrow \{0,1\}$
$(\mathtt{sk}, \mathtt{pk}, \mathtt{evk}) \leftarrow \mathtt{KeyGen}(\lambda)$
$S \leftarrow \emptyset$
$i \leftarrow 0$
$\tilde{q} \leftarrow 0$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}}(\lambda, \mathtt{pk}, \mathtt{evk})$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{evk}}, \mathsf{Dec}'}(\lambda, \mathtt{pk}, \mathtt{evk})$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{evk}}, \mathsf{Dec}_{\mathsf{exact},\mathsf{sk}}}(\lambda, \mathtt{pk}, \mathtt{evk})$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{evk}}, \mathsf{Dec}_{E,\mathsf{sk}}}(\lambda, \mathtt{pk}, \mathtt{evk})$
return $b' = b$
```

```
$\mathsf{Enc}^b_{\mathsf{pk}}(m_0, m_1)$
$\mathtt{ct} \leftarrow \mathtt{Enc}(m_b, \mathtt{pk})$
$\sigma^2_{n(\mathtt{ct})} \leftarrow \left(\frac{1}{6}, \frac{1}{12}\right)$
$S[i] \leftarrow (m_0, m_1, \mathtt{ct}, \sigma^2_{n(\mathtt{ct})})$
$i \leftarrow i + 1$
return $\mathtt{ct}$
```

```
$\mathsf{Eval}^b_{\mathsf{evk}}(g, J = (j_1, \ldots, j_\ell))$
$\mathtt{ct} \leftarrow \mathtt{Eval}(g, S[j_1].\mathtt{ct}, \ldots, S[j_\ell].\mathtt{ct}, \mathtt{evk})$
$\sigma^2_{n(\mathtt{ct})} \leftarrow g(S[j_1].\sigma^2_{n(\mathtt{ct})}, \ldots, S.[j_\ell].\sigma^2_{n(\mathtt{ct})})$
$g_{m_0} \leftarrow g(S[j_1].m_0, \ldots, S[j_\ell].m_0)$
$g_{m_1} \leftarrow g(S[j_1].m_1, \ldots, S[j_\ell].m_1)$
$S[i] \leftarrow (g_{m_0}, g_{m_1}, \mathtt{ct}, \sigma^2_{n(\mathtt{ct})})$
$i \leftarrow i + 1$
return $\mathtt{ct}$
```

```
$\mathsf{Dec}'^b(i)$
If $S[i].m_0 = S[i].m_1 \wedge \tilde{q} \le q$ :
    $B \leftarrow \alpha\sqrt{2n}\sqrt{\sigma^2_{n(\mathtt{ct})[0]} + N\sigma^2_{n(\mathtt{ct})[1]}\sigma^2_s}$
    $\tilde{q} \leftarrow \tilde{q} + 1$
    $\Delta' \leftarrow 2^{\lceil \log(B) \rceil + 1}$
    return $\Delta' \left\lceil \frac{1}{\Delta'} S[i].m_b \right\rfloor$
Else:
    return $\perp$
```

```
$\mathsf{Dec}^b_{\mathsf{exact},\mathsf{sk}}(i)$
If $S[i].m_0 = S[i].m_1$ and $\tilde{q} \le q$ :
    $B \leftarrow \alpha\sqrt{2n}\sqrt{\sigma^2_{S[i].n(\mathtt{ct})[0]} + N\sigma^2_{S[i].n(\mathtt{ct})[1]}\sigma^2_s}$
    $\Delta' \leftarrow 2^{\lceil \log B \rceil + 1}$
    $\tilde{q} \leftarrow \tilde{q} + 1$
    If $||\mathtt{Decode}([\langle S[i].\mathtt{ct}, \mathtt{sk}\rangle]_Q) - S[i].m_b||_\infty \le B$ :
        return $\mathtt{Dec}_r(S[i].\mathtt{ct}, \mathtt{sk})$
    Else
        $\mathsf{BAD} \leftarrow \mathtt{true}$
        return $\mathtt{Dec}_r(S[i].\mathtt{ct}, \mathtt{sk})$
Else:
    return $\perp$
```

```
$\mathsf{Dec}^b_{\mathsf{E},\mathsf{sk}}(i)$
If $S[i].m_0 = S[i].m_1$ and $\tilde{q} \le q$ :
    $B \leftarrow \alpha\sqrt{2n}\sqrt{\sigma^2_{S[i].n(\mathtt{ct})[0]} + N\sigma^2_{S[i].n(\mathtt{ct})[1]}\sigma^2_s}$
    $\Delta' \leftarrow 2^{\lceil \log B \rceil + 1}$
    $\tilde{q} \leftarrow \tilde{q} + 1$
    If $||\mathtt{Decode}([\langle S[i].\mathtt{ct}, \mathtt{sk}\rangle]_Q) - S[i].m_b||_\infty \le B$ :
        return $\mathtt{Dec}_{\delta,r}(S[i].\mathtt{ct}, \mathtt{sk})$
    Else
        $\mathsf{BAD} \leftarrow \mathtt{true}$
        return $\mathtt{Dec}_{\delta,r}(S[i].\mathtt{ct}, \mathtt{sk})$
Else:
    return $\perp$
```

Fig. 3: Games and Oracles for the proof of Theorem 1.

$$\mathtt{Dec}_{\delta,r}(\mathtt{ct}, \mathtt{sk}) = \Delta' \left\lceil \frac{1}{\Delta'} \mathtt{Decode}([\langle S[i].\mathtt{ct}, \mathtt{sk}\rangle]_Q) \right\rfloor = \Delta' \left\lceil \frac{1}{\Delta'} (S[i].m_b + e) \right\rfloor$$

$$= \Delta' \left\lceil \frac{1}{\Delta'} S[i].m_b \right\rfloor = m_r 2^r + \ldots + m_{r+\lceil \Delta' \rceil} 2^{r+\lceil \Delta' \rceil}$$
$$= \mathtt{Dec}_r(\mathtt{ct}, \mathtt{sk}),$$

where the third step holds true since $\frac{1}{\Delta'}||e||_\infty \leq \frac{B}{\Delta'} \leq \frac{2^{\lceil \log(B) \rceil}}{2^{\lceil \log(B) \rceil + 1}} = \frac{1}{2}$, and thus e does not change the value to which $\frac{1}{\Delta'} m$ is rounded, since due to its Δ'-correctability, m does not induce any additional rounding error. Therefore, if the bounds hold, the games $\mathcal{G}_2$ and $\mathcal{G}_3$ are identical. Should the bounds not hold, $\mathtt{Dec}_r(\mathtt{ct}, \mathtt{sk})$ and $\mathtt{Dec}_{\delta,r}(\mathtt{ct}, \mathtt{sk})$ are no longer guaranteed to return the same results. The Games $\mathcal{G}_2$ and $\mathcal{G}_3$ are therefore identical-until-bad-games. Thus we have by the Difference Lemma 5 for their advantage

$$\mathsf{Adv}(\mathcal{G}_2, \mathcal{G}_3) \leq \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_2}) = \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_3}).$$

The probability of the flag BAD being set to true is exactly the probability of the bounds failing. The bounds hold for each call to the decryption oracle with probability $\delta = \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2n}$. They therefore hold during all q calls to the decryption oracle with probability δ^q, and thus fail at least once with probability

$$\mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_2}) = \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_3}) = 1 - \delta^q.$$

Piecing all of the above together, we obtain

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{q-IND-CPA-D}}_{(\delta,r)\text{-exact CKKS}}(\mathcal{A}) \\
&= \left| \mathsf{Pr}\left(\mathsf{q-IND-CPA-D}^{\mathcal{A}} \Rightarrow 1 | \mathsf{q-IND-CPA-D}^{\mathcal{A}} \not\Rightarrow \perp\right) - \frac{1}{2} \right| \\
&= \left| \mathsf{Pr}\left(\mathcal{G}_3^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_3^{\mathcal{A}} \not\Rightarrow \perp\right) - \frac{1}{2} \right| \\
&= \left| \mathsf{Pr}\left(\mathcal{G}_3^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_3^{\mathcal{A}} \not\Rightarrow \perp\right) + \mathsf{Pr}\left(\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_2^{\mathcal{A}} \not\Rightarrow \perp\right) - \mathsf{Pr}\left(\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_2^{\mathcal{A}} \not\Rightarrow \perp\right) - \frac{1}{2} \right| \\
&\leq \left| \mathsf{Pr}\left(\mathcal{G}_3^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_3^{\mathcal{A}} \not\Rightarrow \perp\right) - \mathsf{Pr}\left(\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_2^{\mathcal{A}} \not\Rightarrow \perp\right) \right| + \left| \mathsf{Pr}\left(\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1 | \mathcal{G}_2^{\mathcal{A}} \not\Rightarrow \perp\right) - \frac{1}{2} \right| \\
&= \mathsf{Adv}(\mathcal{G}_2^{\mathcal{A}}, \mathcal{G}_3^{\mathcal{A}}) + \mathsf{Adv}^{\mathcal{G}_2}(\mathcal{A}) \leq \mathsf{Pr}(\mathsf{BAD}_{\mathcal{G}_3}) + \mathsf{Adv}^{\mathcal{G}_2}(\mathcal{A}) \\
&= 1 - \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2nq} + \mathsf{Adv}^{\mathcal{G}_2}(\mathcal{A}) = 1 - \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^{2nq} + \mathsf{Adv}^{\mathcal{G}_1}(\mathcal{A}) \\
&= 1 - \delta^q + \mathsf{Adv}^{\mathcal{G}_0}(\mathcal{A}) = 1 - \delta^q + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{CKKS}}(\mathcal{B}),
\end{aligned}$$

which concludes the proof of the theorem. □

6 Discussion

In this paper we presented a construction for (δ, r)-exact CKKS and proved it is IND-CPA-D secure, as long as the class of admissible circuits is reduced to the class of correctable circuits. Our (δ, r)-exact CKKS construction is similar to "plain" CKKS[7]. In terms of efficiency, the only difference in the parametrization between (δ, r)-exact CKKS and plain CKKS lies in the chosen plaintext scale Δ. Choosing the plaintext scale requires bounds on the noise, which determine how many bits of the plaintext will be polluted by the noise. To obtain a result that has a targeted precision Δ_{target}, the scale Δ needs to be chosen as $\Delta = \Delta_{\mathsf{target}} + B$, where B is a bound on the noise. The tighter the noise bounds, the smaller the Δ, the smaller the overall parameters. Tighter (average-case) bounds come with larger failure probabilities. A larger failure probability of the noise bounds is acceptable in plain CKKS, since it affects correctness but not security. If the noise bound is calculated using average-case analysis, then it is of the form $B = \alpha\sqrt{\sigma^2_{n(\mathtt{ct})[0]} + N\sigma^2_{n(\mathtt{ct})[1]}\sigma^2_s}$, where α determines the tightness of the bound, as well as the failure probability δ. For example a value of $\alpha = 10$ leads to a failure probability of $1 - \delta = 1 - \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^N = 2^{-68}$ for $N = 8192$. For correctness, this failure probability may be acceptable.

For q-IND-CPA-D security, however, we need it to be negligible. Thus, we need to choose a value of $\alpha \geq 14$, since we then have $1 - \mathtt{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^N \leq 2^{-128}$ for $N \leq 65536$. We give an example of how this affects parameters: take Experiment I. (High Degree Chebyshev Power) of Section 4.3, with the circuit $C(X) = T_{4096}(x)$, the 4096-th power of a Chebyshev polynomial of the first kind, with input values uniformly distributed in the interval $[-1, 1]$. This circuit is evaluated with 12 successive evaluations of $y = 2x^2 - 1$. With $\log(N) = 16$, $h = 2N/3$ and $\log(\Delta) = 45$ we achieve a (predicted and experimental) precision of 20.19 bits with a standard deviation of the error (with respect to the expected message) of 2.48 bits. Thus, if we wanted to achieve $\lambda = 128$ q-IND-CPA-D security while keeping the original 20.19 bits of precision and with $q = 2^{15}$, we would have to increase the scale from $\log(\Delta) = 45$ to $\log(\Delta^\star) = \lceil 45 + \log(15) + 2.48 \rceil = 52$, where we took $\alpha = 15$ (see Table 2).

[7] In this Section, we use the term "plain CKKS" to refer to the unmodified version of CKKS [14, 13].

λ	q	n	α	λ	q	n	α
80	1	4096	12	128	1	4096	14
		8192	12			8192	14
		16384	12			16384	14
		32768	12			32768	14
		65536	12			65536	14
	2^5	4096	12		2^5	4096	14
		8192	12			8192	15
		16384	12			16384	15
		32768	12			32768	15
		65536	12			65536	15
	2^{15}	4096	12		2^{15}	4096	15
		8192	13			8192	15
		16384	13			16384	15
		32768	13			32768	15
		65536	13			65536	15

Table 2: Parameter sets leading to an advantage smaller than $2^{-\lambda}$.

Our construction only works if we limit the admissible circuits to correctable ones. Determining the class of correctable circuits exhaustively is outside of the scope of this paper: this is an open question with application to other lattice-based constructions, and we think it is of independent interest. However, we give an example to illustrate the impact of this restriction. A circuit is correctable if the result of its evaluation on plaintexts is in $\frac{1}{\Delta}\mathbb{Z}[i]$. This should always be possible to achieve through scaling, since Δ can be calculated offline by applying our noise bounds. Take the example of the circuit $g(x_1, \ldots, x_n) = x_1 + \ldots + x_n$. Then g is correctable if, for all inputs x_i, we have that $g(x_1, \ldots, x_n)$ is in $\frac{1}{\Delta}\mathbb{Z}[i]$. I.e., if $g(x_1, \ldots, x_n) = x_1 + \ldots + x_n = \Delta(a + bi)$, where a, b are integers. This can be achieved by scaling the inputs x_i by an additional scaling factor Δ' and truncating them to an integer. We conjecture that similar methods for turning any circuit into a correctable circuit exist through scaling, but this needs further investigation.

Acknowledgements

The authors would like to thank Tjerand Silde for feedback on earlier versions of this manuscript.

References

[1] Lattigo v5. Online: `https://github.com/tuneinsight/lattigo`, November 2023. EPFL-LDS, Tune Insight SA.

[2] Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring fhe for practical use. Cryptology ePrint Archive, Paper 2024/203, 2024. `https://eprint.iacr.org/2024/203`.

[3] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. In *Advances in Cryptology - Eurocrypt 2004*, 2004.

[4] Mariya Georgieva Belorgey, Sergiu Carpov, Nicolas Gama, Sandra Guasch, and Dimitar Jetchev. Revisiting key decomposition techniques for fhe: Simpler, faster and more generic. Cryptology ePrint Archive, Paper 2023/771, 2023. `https://eprint.iacr.org/2023/771`.

[5] 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.

[6] 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.

[7] 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.

[8] Marina Checri, Renaud Sirdey, Aymen Boudguiga, Jean-Paul Bultel, and Antoine Choffrut. On the practical CPAD security of "exact" and threshold FHE schemes and libraries. Cryptology ePrint Archive, Paper 2024/116, 2024. `https://eprint.iacr.org/2024/116`.

[9] 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.

[10] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA-D security of exact FHE schemes. Cryptology ePrint Archive, Paper 2024/127, 2024. `https://eprint.iacr.org/2024/127`.

[11] 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`.

[12] 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.

[13] 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.

[14] 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.

[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1), 2020.

[16] 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.

[17] 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.

[18] Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. To appear in the post-proceedings of Selected Areas of Cryptography (SAC), 2023.

[19] 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.

[20] Anamaria Costache, Lea Nürnberger, and Rachel Player. Optimisations and trade-offs for HElib. In Mike Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, pages 29–53, Cham, 2023. Springer International Publishing.

[21] N. Drucker, G. Moshkowich, T. Pelleg, and H. Shaul. BLEACH: Cleaning errors in discrete computations. *Journal of Cryptology*, 2024.

[22] 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.

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

[24] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*. Springer, 2013.

[25] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures, 2024. To be pulished at 33rd USENIX Security Symposium.

[26] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020*, 2020.

[27] Ilia Iliashenko. Optimisations of fully homomorphic encryption. 2019.

[28] Charanjit S. Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. Springer-Verlag, 2022.

[29] Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. EvalRound algorithm in CKKS bootstrapping. Springer-Verlag, 2022.

[30] 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.

[31] 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.

[32] Yongwoo Lee, Joonwoo Lee, Young-Sik Kim, and Jong-Seon No. Near-optimal polynomial for modulus reduction using l2-norm for approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2020/488, 2020. `https://eprint.iacr.org/2020/488`.

[33] 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.

[34] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, Cham, 2022. Springer Nature Switzerland.

[35] Sean Murphy and Rachel Player. A central limit framework for Ring-LWE decryption. Cryptology ePrint Archive, Paper 2019/452, 2019. `https://eprint.iacr.org/2019/452`.

[36] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

A The RNS-CKKS scheme

We present the RNS version [13] of the CKKS [14] scheme. We present it similarly to [5]. We do not give the algorithms for Rotations, Conjugations and Matrix-ciphertext products here, since we base this paper on more efficient versions of those algorithms as they have been given in [5]. We therefore present them in Section 3.

A.1 RNS-CKKS Bootstrapping

We give here a high level overview for CKKS bootstrapping ([9],[12],[26],[31]) and its RNS variant ([5],[31]).

The goal in CKKS is to transform a ciphertext at the lowest level to a ciphertext at a higher level. Before bootstrapping, CKKS ciphertexts are assumed to have a low noise and are transformed into ciphertexts with respect to a bigger ciphertext modulus, but with potentially more noise.

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 $\mathtt{pt}$. Note that the encoding format of $\mathtt{pt}$ is irrelevant to the bootstrapping circuit as the aim of the bootstrapping is to increase the level of the ciphertext without changing the underlying plaintext.

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

$\texttt{SecKeyGen}(\lambda)$	Sample $s \leftarrow \mathcal{S}$, where $\mathcal{S}$ is a distribution over $\mathcal{R}_{PQ_L}$ producing elements of small norm, where L is the multiplicative depth. Return $\texttt{sk} := (1, s)$.
$\texttt{PubKeyGen}(\lambda, s)$	Sample $a \overset{\$}{\leftarrow} \mathcal{R}_{PQ_L}, e \leftarrow \chi$, where χ is some distribution over $\mathcal{R}_{PQ_L}$ with standard deviation σ_0. Return $\texttt{pk} := ([as+e]_{PQ_L}, [a]_{PQ_L})$.
$\texttt{EvalKeyGen}(\lambda, s, s')$	Let $\mathbf{w}$ be an integer decomposition base with β elements, designed as follows $\mathbf{w}_i = \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right)^{-1} \right]_{q_{\gamma_i}}$, where $q_{\gamma_i} = \prod_{j=\gamma i}^{\min(\gamma(i+1)-1, L)} q_j$, for $0 \le i < \beta := \left\lceil \frac{L+1}{\gamma} \right\rceil$, and γ a positive integer. Let P be a modulus and sample $a_i \overset{\$}{\leftarrow} \mathcal{R}_{PQ_L}$, and $e_i \leftarrow \chi$ for $i \in \{0, \ldots, \beta-1\}$. Compute $$\left(\texttt{evk}^0_{q_{\gamma_i}, s \to s'}, \texttt{evk}^1_{q_{\gamma_i}, s \to s'}\right) = \left([-a_i s' + sP \frac{Q_L}{q_{\gamma_i}} \left[\left(\frac{Q_L}{q_{\gamma_i}} \right)^{-1} \right]_{q_{\gamma_i}} + e_i]_{PQ_L}, [a_i]_{PQ_L} \right) \forall i.$$ $\texttt{evk}$ is needed for switching a ciphertext from being with respect to a secret key component s to being with respect to a secret key component s'. Return $\texttt{evk}$.
$\texttt{CtS}(\mathbf{m} \in \mathbb{C}^n, \varDelta)$ (Encode a message $\mathbf{m}$ from coefficients to slots)	Let n be such that $n \vert N$. Compute $\mathbf{m}' = \texttt{DFT}_n^{-1}(\mathbf{m})$. Compute $\mathbf{m}'_0 = \left\lceil \varDelta \frac{1}{2}(\mathbf{m}' + \overline{\mathbf{m}'}) \right\rfloor, \mathbf{m}'_1 = \left\lceil \varDelta \frac{-i}{2}(\mathbf{m}' - \overline{\mathbf{m}'}) \right\rfloor$. Return $\texttt{pt} = \mathbf{m}'_0 \vert\vert \mathbf{m}'_1 \in \mathcal{R}$.
$\texttt{StC}(\texttt{pt} \in \mathcal{R}, \varDelta)$ (Decode a plaintext $\texttt{pt}$ from Slots to Coefficients)	Let $\texttt{pt} = \mathbf{m}'_0 \vert\vert \mathbf{m}'_1$. Return $\mathbf{m} = \texttt{DFT}_n(\varDelta^{-1}(\mathbf{m}'_0 + i \cdot \mathbf{m}'_1))$.
$\texttt{Enc}(\texttt{pt}, \texttt{sk}, \ell)$ (Secret key encryption)	Sample $a \overset{\$}{\leftarrow} \mathcal{R}_{Q_\ell}$, and $e \leftarrow \chi$. Return $\texttt{ct} = ([-as + e + \texttt{pt}]_{Q_\ell}, a)$.
$\texttt{Enc}(\texttt{pt}, \texttt{pk}, \ell)$ (Public key encryption)	Sample $u \leftarrow \mathcal{S}, e^{(0)}, e^{(1)} \leftarrow \chi$. Compute $d = (\texttt{pk}[0] \cdot u + e^{(0)}, \texttt{pk}[1] \cdot u + e^{(1)}) \in \mathcal{R}_{PQ_\ell}$ and return $\texttt{ct} = (\texttt{pt}, 0) + \left\lceil P^{-1} \cdot d \right\rfloor \in \mathcal{R}_{Q_\ell}$.
$\texttt{Dec}(\texttt{ct}, \texttt{sk})$	Return $\texttt{pt} = \texttt{ct}(\texttt{sk}) = \sum_i \texttt{ct}[i]\texttt{sk}^i$.

Fig. 4: Overview over the RNS-CKKS scheme 1

$\texttt{KeySwitch}(\{\texttt{poly}, Q_\ell, \varDelta\}, s, s', \texttt{evk}_{s' \to s})$	Decompose $\texttt{poly}$ base $\mathbf{w}$ into $\mathbf{p}$, such that $\texttt{poly} = \langle \mathbf{p}, \mathbf{w} \rangle$ and return $\lceil P^{-1} \langle \mathbf{p}, \texttt{evk}_{s' \to s} \rfloor \rangle$ $\texttt{Add}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \varDelta_0\}, \{\texttt{ct}^{(1)}, Q_{\ell_1}, \varDelta_1\})$. Scale $\texttt{ct}^{(0)}$, $\texttt{ct}^{(1)}$ to $\max(\varDelta_0, \varDelta_1)$, and return $\{\texttt{ct}^{(0)} + \texttt{ct}^{(1)}, \min(Q_{\ell_0, Q_{\ell_1}}, \max(\varDelta_0, \varDelta_1))\}$.
$\texttt{AddConst}(\{\texttt{ct}, Q_\ell, \varDelta\}, \texttt{const} = a + bi)$	Return $\{\texttt{ct}' = ([\texttt{ct}[0] + \left\lceil \varDelta(a + bX^{\frac{n}{2}}) \right\rfloor]_{Q_\ell}, \texttt{ct}[1]), Q_\ell, \varDelta\}$.
$\texttt{AddPlain}(\{\texttt{ct}, Q_\ell, \varDelta\}, \{\texttt{pt}, Q_{\ell'}, \varDelta'\})$	Scale $\texttt{ct}$ and $\texttt{pt}$to $\max(\varDelta, \varDelta')$ and return $\{\texttt{ct}' = ([\texttt{ct}[0] + \texttt{pt}]_{Q_\ell}, \texttt{ct}[1]), \min(Q_\ell, Q_{\ell'}, \max(\varDelta, \varDelta'))\}$.
$\texttt{MultConst}(\{\texttt{ct}, Q_\ell, \varDelta\}, \texttt{const} = a + bi, \varDelta')$	Return $\{\lceil \varDelta' a \rfloor \, \texttt{ct} + \lceil \varDelta' b \rfloor \, \texttt{ct} X^{\frac{n}{2}}, Q_\ell, \varDelta\varDelta'\}$.
$\texttt{MultPlain}(\{\texttt{ct}, Q_\ell, \varDelta\}, \{\texttt{pt}, Q_{\ell'}, \varDelta'\})$	Return $\{\texttt{ct}' = ([\texttt{ct}[0]\texttt{pt}]_{Q_i}, [\texttt{ct}[1]\texttt{pt}]_{Q_i}, \min(Q_\ell, Q_{\ell'}), \varDelta\varDelta'\})$.
$\texttt{Tensor}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \varDelta_0\}, \{\texttt{ct}^{(1)}, Q_{\ell_1}, \varDelta_1\})$	Compute $\texttt{ct}' = (\texttt{ct}^{(0)}[0]\texttt{ct}^{(1)}[0], \texttt{ct}^{(0)}[1]\texttt{ct}^{(1)}[0] + \texttt{ct}^{(0)}[0]\texttt{ct}^{(1)}[1], \texttt{ct}^{(0)}[1]\texttt{ct}^{(1)}[1])$. Return $\texttt{ct}'$.
$\texttt{Relinearization}(\{\texttt{ct} = (\texttt{ct}[0], \texttt{ct}[1], \texttt{ct}[2]), Q_\ell, \varDelta\})$	$\texttt{ct}' = (\texttt{ct}'[0], \texttt{ct}'[1]) + \texttt{KeySwitch}(\texttt{ct}[2], \texttt{evk}_{s^2 \to s})$.
$\texttt{Rescale}(\{\texttt{ct}, Q_\ell, \varDelta\})$	Return $\{(\lceil q_\ell^{-1} \texttt{ct}[0] \rfloor, \lceil q_\ell^{-1} \texttt{ct}[1] \rfloor), Q_{\ell-1}, \frac{\varDelta}{q_\ell}\}$.
$\texttt{DropLevel}(\{\texttt{ct}, Q_\ell, \varDelta\}, k)$	Return $\{\texttt{ct}, Q_{\ell-k}, \varDelta\}$.

Fig. 5: Overview over the RNS-CKKS scheme 2

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 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 $(f^2)_i, i \in \{0, \dots, N-1\}$*

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

Proof. Since $f \in \mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, following a result given by [27], 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 $i \neq 0$, we have the same variances of the terms as above. Before we rewrote the sum, there were $N+1$ additive terms. We have pulled the term $(f_{\frac{N}{2}})^2$ out. Since each of the remaining terms appeared twice, 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 (Rescaling). s*Let* $\{\texttt{ct}, Q_\ell, \Delta\}$ *be a ciphertext encrypting a plaintext* $\texttt{pt}$ *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*

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

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

Proof. The noise estimate for $\texttt{ct}^{(1)}$ follows trivially from the definition of $\texttt{DropLevel}$ as given in Table 5. 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 $n(\texttt{ct}^{(2)}) = q_\ell^{-1}n(\texttt{ct}) + (\tau^{(0)}, \tau^{(1)})$. It is then easy to calculate the coefficient standard deviations of $n(\texttt{ct}^{(2)})$ as

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

☐

Lemma 7 (Fresh Encryption) *Let* $\mathbf{m}$ *be a message with scaling factor* Δ, *and let* $\texttt{pt} \in \mathcal{R}_{Q_\ell}$ *be the encoding of* $\mathbf{m}$. *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*

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

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 $\left[\frac{1}{2}, \frac{1}{2}\right)$.

A public-key ciphertext is created as follows: first, a randomized encryption of 0 is sampled with respect to a special modulus QP using $\texttt{pk}$. Here, Q is the ciphertext modulus corresponding to the level of the ciphertext is encrypted, and P is an auxiliary modulus. The ciphertext is then rescaled by P, and the output is a ciphertext with respect to the ciphertext modulus Q. Finally, the plaintext is added to the first component. Thus we have for the structure of a freshly encrypted ciphertext

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

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

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

as long as $||(e_{\texttt{pk}}u + e^{(0)}, e^{(1)})||_\infty \leq P/2$, which is the case with high probability because $e_{\texttt{pk}}, e^{(0)}, e^{(1)}$ 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

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

The secret key ciphertext is created even more easily as $\texttt{ct}_{\texttt{sk}} = (-as + e, a)$, 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 8 (Addition by Plaintext). *Let* $\{\texttt{ct}^{(2)}, Q_{\ell_2}, \Delta_2)\} = \texttt{AddPlain}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\texttt{pt} + n(\texttt{pt}), Q, \Delta\})$ *be the result of a plaintext-ciphertext addition. Then we have for the standard deviation of the component-wise noise of* $\texttt{ct}^{(2)}$*, and for the standard deviation of a plaintext-plaintext addition*

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

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 $n(\mathtt{ct}^{(2)})$

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

□

Lemma 9 (Addition by Constant). *Let* $\mathtt{const} \in \mathbb{C}^n$ *be a constant. Let* $\{\mathtt{ct}^{(2)}, Q_{\ell_2}, \Delta_2\} = \mathtt{AddConst}(\{\mathtt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \mathsf{const})$ *the result of a constant-ciphertext addition. Then we have for the standard deviation of the component-wise noise*

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

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 11 (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* $\{\mathtt{ct}^{(2)}, Q_{\ell_2}, \Delta_2\} =$

$\texttt{MultPlain}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \{\texttt{pt} + n(\texttt{pt}), Q, \Delta\})$). Then we have for the coefficient standard deviation of the component-wise noise

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

Proof. If we calculate a plaintext-plaintext product we can easily read of the noise of the resulting plaintext.

$$\begin{aligned}\texttt{pt}_\times + n(\texttt{pt}_\times) &= (\texttt{pt}^{(0)} + n(\texttt{pt}^{(0)})(\texttt{pt}^{(1)} + n(\texttt{pt}^{(1)}))\\ &= \texttt{pt}^{(0)}\texttt{pt}^{(1)} + n(\texttt{pt}^{(0)})\texttt{pt}^{(1)} + n(\texttt{pt}^{(1)})\texttt{pt}^{(0)} + n(\texttt{pt}^{(0)})n(\texttt{pt}^{(1)}),\end{aligned}$$

and therefore we have that $n(\texttt{pt}_\times) = n(\texttt{pt}^{(0)})\texttt{pt}^{(1)} + n(\texttt{pt}^{(1)})\texttt{pt}^{(0)} + n(\texttt{pt}^{(0)})n(\texttt{pt}^{(1)})$.

Thus, we have for the coefficient standard deviation of $n(\texttt{pt}_\times)$

$$\begin{aligned}\sigma_{n(\texttt{pt}_\times)_i} &= \sigma_{(n(\texttt{pt}^{(0)})\texttt{pt}^{(1)}) + (n(\texttt{pt}^{(1)})\texttt{pt}^{(0)}) + (n(\texttt{pt}^{(0)})n(\texttt{pt}^{(1)}))}\\ &= \sqrt{\sigma^2_{(n(\texttt{pt}^{(0)})\texttt{pt}^{(1)})} + \sigma^2_{(n(\texttt{pt}^{(1)})\texttt{pt}^{(0)})} + \sigma^2_{(n(\texttt{pt}^{(0)})n(\texttt{pt}^{(1)}))}}\\ &= \sqrt{N(\sigma^2_{n(\texttt{pt}^{(0)})}\sigma^2_{\texttt{pt}^{(1)}} + \sigma^2_{n(\texttt{pt}^{(1)})}\sigma^2_{\texttt{pt}^{(0)}} + \sigma^2_{n(\texttt{pt}^{(0)})}\sigma^2_{n(\texttt{pt}^{(1)})})}.\end{aligned}$$

The plaintext-ciphertext product is computed as $\{\texttt{ct}^{(2)}, Q_{\ell_2}, \Delta_2\} = \{(\texttt{ct}^{(0)}[0](\texttt{pt} + n(\texttt{pt})), \texttt{ct}^{(0)}[1](\texttt{pt} + n(\texttt{pt}))), \min(Q_{\ell_0}, Q), \Delta_0\Delta\}$. Therefore, the ciphertext noise gets multiplied by both the plaintext and the plaintext noise, and both become a part of the resulting noise. That is, we have

$$\begin{aligned}n(\texttt{ct}^{(2)})[i] &= n(\texttt{ct}^{(0)}[i]\texttt{pt})\\ &= n(\texttt{ct}^{(0)})[i]\texttt{pt} + n(\texttt{ct}^{(0)})[i]n(\texttt{pt}),\end{aligned}$$

$i \in \{0, 1\}$, and then the result follows by Lemma 1. □

Lemma 12 (Multiplication by Constant). *Let $\texttt{const} = a + bi \in \mathbb{C}^n$ be a constant. Let $\{\texttt{ct}^{(2)}, Q_{\ell_0}, \Delta_0 q_{\ell_0}\} = \texttt{MultConst}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, \texttt{const}, q_{\ell_0})$. Then we have for the coefficient standard deviation*

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

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

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

$j \in \{0,1\}$ 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 13 (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*

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

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

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

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

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

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

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

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

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

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

where the last line follows from the component-wise noise consisting of all the terms that do no have the mask a, which will be cancelled out during decryption.

Thus we get for the coefficients variances

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

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

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

Lastly, for squaring we can calculate the component-wise noise similarly, using the same way of denoting $\mathtt{ct}^{(0)}$. We get

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

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

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

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

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

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

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

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

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

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

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

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 parts of the ciphertext are being cancelled out, it is easier to calculate the term $\mathtt{ct}[0] + \mathtt{ct}[1]s$ and to retrieve the component-wise noise from the result. We get

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

From this we can easily see that

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

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

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

□

Lemma 15 (Relinearization). *Let $\{\mathtt{ct}, Q_\ell, \Delta\}$ be a ciphertext resulting from* $\mathtt{Tensor}$*. Note that* $\mathtt{ct}^{(0)} = (\mathtt{ct}[0], \mathtt{ct}^{(0)}[1], \mathtt{ct}^{(0)}[2])$*. Let* $\{\mathtt{ct}, Q_\ell, \Delta\} = \mathtt{Relin}(\{\mathtt{ct}, Q_\ell, \Delta_0\}) = (\mathtt{ct}[0], \mathtt{ct}[1]) + \mathtt{KeySwitch}(\mathtt{ct}[2], s^2)$*. Then we have for the coefficient standard deviation of the component-wise noise*

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

Proof. Let
$\mathtt{evk}_{s^2\to s} = ((\mathtt{evk}_{s^2\to s}[0][0], \dots, \mathtt{evk}_{s^2\to s}[\beta-1][0]),$
$(\mathtt{evk}_{s^2\to s}[0][1], \dots, \mathtt{evk}_{s^2\to s}[\beta-1][1]))$ 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 of [5] the ciphertext $\mathtt{ct}$ is calculated as

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

To determine which terms get cancelled out during decryption, and therefore which terms are part of the component-wise noise, we calculate what the decryption of $\mathtt{ct}$ minus $\mathtt{pt}^{(0)}$ looks like

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

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

Therefore, we have for the component-wise noise

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

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

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

□

Lemma 16 (Rotations). *Let* $\{\texttt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}$ *be a ciphertext encrypting* $\texttt{pt}^{(0)}$ *with respect to a secret key* $\texttt{sk} = (1, s)$. *Let* $\{\texttt{ct}, Q_{\ell_0}, \Delta_0\} = \texttt{Rotate}(\{\texttt{ct}^{(0)}, Q_{\ell_0}, \Delta_0\}, k)$ *be a ciphertext encrypting* $\Phi_k(m_{\texttt{ct}})$ *with respect to* $\Phi_k(\texttt{sk}) = (1, \Phi_k(s))$. *Then we have for the coefficient standard deviation of the component-wise noise*

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

Proof. The proof is very similar to the proof of Lemma 14. We have as evaluation keys $\texttt{evk}_{s\to\Phi_k^{-1}(s)} = ((\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta][0]),$ $(\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta][1]))$, where

$$\texttt{evk}_{s\to\Phi_k^{-1}(s)}[i][0] = \left[a_i\Phi_k^{-1}(s) + sP\frac{Q_L}{q_{\gamma_i}}\left[\left(\frac{Q_L}{q_{\gamma_i}}\right)^{-1}\right]_{q_{\gamma_i}} + e_i\right]_{PQ_L}$$
$$\texttt{evk}_{s\to\Phi_k^{-1}(s)}[i][1] = [-a_i]_{PQ_L}.$$

By Algorithm 4 in [5] the ciphertext after rotation is then calculated as

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

And therefore,

$$\begin{aligned}\texttt{ct}[0] + \texttt{ct}[1]s - \Phi_k(\texttt{pt}^{(0)}) &= \Phi_k(\texttt{ct}^{(0)}[0] + \lceil P^{-1}\langle([\texttt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\texttt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}})\\ &(\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][0])\rangle\rfloor)\\ &\quad + \Phi_k(\lceil P^{-1}\langle([\texttt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\texttt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}}),\\ &(\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][1])\rangle\rfloor))s - \Phi_k(\texttt{pt}^{(0)})\\ &= \Phi_k(\texttt{ct}^{(0)}[0] + \lceil P^{-1}\langle([\texttt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\texttt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}})\\ &(\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][0], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][0])\rangle\rfloor\\ &\quad + \lceil P^{-1}\langle([\texttt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\texttt{ct}^{(0)}[1]]_{q_{\gamma_{\beta-1}}}),\\ &(\texttt{evk}_{s\to\Phi_k^{-1}(s)}[0][1], \ldots, \texttt{evk}_{s\to\Phi_k^{-1}(s)}[\beta-1][1])\rangle\rfloor\Phi_k^{-1}(s) - \texttt{pt}^{(0)})\\ &= \Phi_k\Bigg(\texttt{ct}^{(0)}[0] + P^{-1}\sum_{j=0}^{\beta-1}[\texttt{ct}^{(0)}[1]]_{q_{\gamma_j}}\end{aligned}$$

$$\Big(\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[j][0]+\mathtt{evk}_{s\to\Phi_k^{-1}(s)}[j][1]\Phi^{-1}(s)\Big)$$
$$-\,\mathtt{pt}^{(0)}+\tau^{(0)}+\tau^{(1)}s\Bigg)$$
$$=\Phi_k\Bigg(\mathtt{ct}^{(0)}[0]+P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}$$
$$\left(sP\frac{Q_L}{q_{\gamma_j}}\left[\left(\frac{Q_L}{q_{\gamma_j}}\right)^{-1}\right]_{q_{\gamma_j}}+e_j\right)+\tau^{(0)}+\tau^{(1)}s-\mathtt{pt}^{(0)}\Bigg)$$
$$=\Phi_k\Bigg(\mathtt{ct}^{(0)}[0]+\mathtt{ct}^{(0)}[1]s-\mathtt{pt}^{(0)}+P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}e_j$$
$$+\,\tau^{(0)}+\tau^{(1)}s\Bigg)$$
$$=\Phi_k(\mathtt{ct}^{(0)}[0]+\mathtt{ct}^{(0)}[1]s-\mathtt{pt}^{(0)})+\Phi_k(P^{-1}\sum_{j=0}^{\beta-1}[\mathtt{ct}^{(0)}[1]]_{q_{\gamma_j}}e_j)\mathtt{ct}$$
$$+\,\Phi_k(\tau^{(0)}+\tau^{(1)}s).$$

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 its noise remains the same. Therefore, the standard deviations of the component-wise noise are the same as in the case of key switching. □

Lemma 17 (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. □

C Noise Standard Deviations for Bootstrapping

C.1 Noise Standard Deviation for Homomorphic En-and Decoding

In this section we show how to build up 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 [5], stated again in Algorithm 3 in Appendix D 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 [5]. 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 [5]. 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.*

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

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

$$\mathtt{ct}_{\mathsf{pre-rot}(i\geq 1)}[0]$$

$$= \varPhi_i(P\mathtt{ct}^{(0)}[0] + \langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),$$

$$(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle)$$

$$= P\varPhi_i(\mathtt{ct}^{(0)}[0] + P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),$$

$$(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle)$$

$$\mathtt{ct}_{\mathsf{pre-rot}(i \geq 1)}[1]$$

$$= \varPhi_i(\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),$$

$$(\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle)$$

$$= P\varPhi_i(P^{-1}\langle([\mathtt{ct}^{(0)}[1]]_{q_{\gamma_0}}, \ldots, [\mathtt{ct}^{(0)}[1]]_{q_{\gamma_\beta}}),$$

$$(\mathtt{evk}_{\mathsf{rot}}[0][1], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][1])\rangle).$$

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 $\mathtt{ct}_{\mathsf{pre-rot}(i)}$ multiplied by $M^{n_1 j+i}$. That is we get

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

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

We get for the ciphertext after the n_2-loop

$$\mathtt{ct}_{n_2-\mathsf{loop}}[0] = \sum_{j=0}^{n_2-1} \varPhi_{n_1 j}(\mathtt{ct}_{n_1-\mathsf{loop},j}[0] + \langle([\lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_0}},$$

$$\ldots, [\lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_{\beta-1}}}), (\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle),$$

$$= \sum_{j=0}^{n_2-1} \varPhi_{n_1 j}(\mathtt{ct}_{n_1-\mathsf{loop},j}[0] + \lceil P^{-1}\langle([\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_0}},$$

$$\ldots, [\mathtt{ct}_{n_1-\mathsf{loop},j}[1]]_{q_{\gamma_\beta}})(\mathtt{evk}_{\mathsf{rot}}[0][0], \ldots, \mathtt{evk}_{\mathsf{rot}}[\beta][0])\rangle\rfloor),$$

$$= \sum_{j=0}^{n_2-1} \mathtt{Rotate}(\mathtt{ct}_{n_1-\mathrm{loop},j})[0].$$

$$\mathtt{ct}_{n_2-\mathsf{loop}}[1] = \sum_{j=0}^{n_2-1} \varPhi_{n_1 j}(\langle([\lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_0}}, \ldots, [\lceil P^{-1}\mathtt{ct}_{n_1-\mathsf{loop},j}[1]\rfloor]_{q_{\gamma_\beta}}),$$

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

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 10 and 16. $\mathtt{ct}$ 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 D.

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 submatrices. Let $E_i := \left\lceil \Delta\tau^{-1}(\left(\frac{1}{2}\right)^{\frac{1}{k}} SF_i^{-1})\right\rfloor$ 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*

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

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

Remark 3. The Special Fourier matrices are decomposed for better efficiency. A description of the decomposition can be found in [11].

Proof. The coefficients to slots operation is calculated following [5] 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 10, 12, 17, and 18. Thus we get

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

$$\sigma_{n(\mathtt{ct}^{(0)})[1]} = \left(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[0]} + \frac{1}{12}\right)^{\frac{1}{2}}$$

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

$$\sigma_{n(\mathtt{ct}^{(2)})[1]} = \left(2\sigma^2_{n(\mathtt{MCtxt}(\ldots,(\mathtt{MCtxt}(\mathtt{ct}^{(0)},E_k),E_{k-1}),\ldots,E_0))[1]} + \frac{1}{12}\right)^{\frac{1}{2}}.$$

□

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:     $\mathtt{ct} \leftarrow \mathtt{MatMul}(\mathtt{ct}, \mathsf{SF}_i)$
4:     $\mathtt{ct} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
5: end for
6: return $\mathtt{ct}$
```

Lemma 20 (Slots to Coefficients). *Let $(\{ct^{(1)}, Q_\ell, \Delta\}, \{ct^{(2)}, Q_\ell, \Delta\})$ be two ciphertexts in slots format, each encoding n real coefficients. Let $\{ct^{(0)}, Q_{\ell-k}, \Delta_0\} = StC(\{ct^{(1)}, Q_\ell, \Delta\}, \{ct^{(2)}, Q_\ell, \Delta\})$ 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*

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

Proof. Following [5], the Slots to Coefficient operation is evaluated as

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

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. □

C.2 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 5, 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 [12, 9, 31, 30, 32, **?**, 28, 29]. We choose the approach of [26] as it provides in practice the best overall performance in term of precision and depth consumption and is the default approach in Lattigo, among the ones proposed. 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 [1] for our analysis and we recall it in Algorithm 6 in Appendix D.

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, as [26] pointed out, when evaluated homomorphically we have a better noise growth when computing the Chebyshev polynomials via 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 $a = b = j/2$ (else it is not depth optimal). 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 they are mathematically equivalent. Since the computation of the Chebyshev polynomials is mostly implemented recursively in practice, giving the noise estimates theoretically non-recursively would lead to a large difference between theory and practice of 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

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

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 $T_i(t)$

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

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, by again modelling the τ_i as being uniform random with variance $\frac{1}{12}$. $\square$

Let $p(t)$ be the a degree d polynomial that we want to evaluate on $\texttt{ct}$, and let $T = \{\{T_0, T_1, \dots, T_{2^{l-1}}\}, \{T_{2^l}, T_{2^{l+1}}, \dots, T_{2^m}\}\}$ 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 $c_i \in \mathbb{C}$, $\mathbf{T} = \{1, \texttt{ct}, T_2(\texttt{ct}), \dots, T_d(\texttt{ct})\}$ 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 8*

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

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 8. 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 10 and 12. $\square$

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* ***GiantStep*** *(Algorithm 9) 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 9*

$$\begin{aligned}
\sigma^2_{n(ct'_1)} &= \sigma^2_{n(\textit{Rescale}(\textit{Relinearize}(ct_1)))} \\
&= \left(q_\ell^{-2}\left(\sigma^2_{n(ct_1[0])} + \sigma^2_{\textit{ks-add}}\right) + \frac{1}{12}, q_\ell^{-2}\left(\sigma^2_{n(ct_0[1])} + \frac{1}{12}\right) + \frac{1}{12}\right) \\
\sigma_{n(\textit{GiantStep}(ct_0,ct_1,ct_2))} &= \left(\sqrt{\sigma^2_{n(ct_0[0])} + N\left(\sigma^2_{pt_1} + \sigma^2_{n(ct'_1[0])}\right)\left(\sigma^2_{pt_2} + \sigma^2_{n(ct_2[0])}\right)}, \right. \\
&\quad \sqrt{\sigma^2_{n(ct_0)[1]} + N\left(\sigma^2_{n(ct_2)[1]}\left(\sigma^2_{pt_1} + \sigma^2_{n(ct_1)[0]}\right) + \sigma^2_{n(ct_1)[1]}\left(\sigma^2_{pt_2} + \sigma^2_{n(ct_2)[0]}\right)\right)}, \\
&\quad \left. \sqrt{N\sigma^2_{n(ct_1)[1]}\sigma^2_{n(ct_2)[1]}} \right)
\end{aligned}$$

Proof. The proof follows directly from Lemmas 6,10,13,15. $\square$

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

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

where

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

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

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

□

C.3 Noise Standard Deviation for the Homomorphic Modulus Reduction

Lastly, we give an expression for the standard deviaiton of the component-wise noise after homomorphic modulus reduction, which is the last building block that remains to get the standard deviation and therefore a noise estimate of the bootstrapping circuit.

Lemma 25 (EvalMod1). *Let $\{\texttt{ct}, Q_{\ell+1}, \Delta\}$ 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 $\texttt{ct}_{\mathsf{final}} = \texttt{EvalMod1}(\{\texttt{ct}, Q_{\ell-r}, \Delta\}, p(t), K)$ be the result of Algorithm 5. Let $\texttt{ct}^{(j)}$ be the result after the $j-$th iteration of the loop in lines $10 - -16$ in Algorithm 5. Let $\{\texttt{ct}', Q_{\ell-\lceil \log(d) \rceil}, \Delta_r\} = \texttt{EvaluatePolynomial}(\{\texttt{ct}'', Q_\ell, \Delta\}, p(t))$, and $\{\texttt{ct}'', Q_\ell, \Delta\} = \texttt{AddConst}(\texttt{Rescale}(\texttt{MultConst}(\{\texttt{ct}, Q_{\ell+1}, \Delta\}, 2)), -\left(\frac{-0.25}{2^r K}\right))$. Then we have for the coefficient standard deviation of the component-wise noise of $\texttt{ct}_{\mathsf{final}}$*

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

where

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

for $0 < j \le r - 1$, and

$$\sigma_{n(\texttt{ct}^{(0)})} = \sqrt{4\sigma^{\mathbf{2}}{}_{n(\texttt{Relin}(\texttt{Tensor}(\texttt{ct}', \texttt{ct}')))} + \left(\frac{q_{\ell-1}^{-2}+1}{12}, \frac{q_{\ell-1}^{-2}+1}{12}\right)}$$

$$\sigma_{n(ct'')} = \sigma_{n(\textit{EvaluatePolynomial}(ct'',p))}$$
$$\sigma_{n(ct')} = \sqrt{2^{\frac{2\lceil \log(q_0)\rfloor}{q_0}}\sigma^{\mathbf{2}}{}_{n(ct)} + \left(\frac{q_\ell^{-2}+1}{12}, \frac{q_\ell^{-2}+1}{12}\right)}.$$

Proof. We build the noise estimates up from bottom to top. First we see that

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

Thus,

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

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

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

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

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

from which the claimed standard deviation easily follows. □

C.4 Noise Standard Deviation for Bootstrapping

We now finally have all the ingredients that we need to give a noise estimate for bootstrapping. RNS-CKKS bootstrapping consists of taking the ciphertext with respect to a bigger modulus Q, then applying the coefficients to slots transformation, then evaluating the sine function on the ciphertext that is approximating the reduction modulo Q and is in turn being approximated by a polynomial interpolation, and lastly applying the slots to coefficients transformation. The mod-up operation does not introduce extra noise, since the ciphertext is not scaled up, but merely considered with respect to a higher modulus. We therefore easily can see the following Lemma.

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. □

D Algorithms

Algorithm 2 Optimized Hoisting Rotations (Algorithm 4 in [5])

```
Require: $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) \in \mathcal{R}_{Q_\ell}^2$ and a set of rotation keys $\mathtt{evk}_{s \to \Phi_k^{-1}(s)}$.
Ensure: $\mathbf{v}$ a list containing each $k$ rotation of $\mathtt{ct}$.
1: $\mathbf{d} \leftarrow [[\mathtt{ct}[1]_{q_{\gamma_{0 \le i < \beta}}}]]_{PQ_\ell}$                ▷ Decompose
2: for all $k$ do
3:     $(a, b) \leftarrow (\langle \mathbf{d}, \mathsf{evk}^{(0)}_{s \to \Phi_k^{-1}(s)} \rangle, \langle \mathbf{d}, \mathsf{evk}^{(1)}_{s \to \Phi_k^{-1}(s)} \rangle)$
4:     $(a, b) \leftarrow (\lceil P^{-1} a \rfloor \lceil P^{-1} b \rfloor)$
5:     $\mathbf{v}_k \leftarrow (\Phi_k(\mathtt{ct}[0] + a, \Phi_k(b)))$
6: end for
7: return $\mathbf{v}$
```

Algorithm 3 Double-hoisting BSGS matrix-ciphertext algorithm (Algorithm 6 in [5])

```
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: $\mathbf{d} \leftarrow [[\mathtt{ct}[1]_{q_{\gamma_i}}]_{0\le i<\beta}]_{PQ_i}$
 2: $(a^{(0)}, b^{(0)}) \leftarrow (P\mathtt{ct}[0], P\mathtt{ct}[1])$
 3: for $i = 1; i < n_1; i++$ do
 4:     $a^{(i)} \leftarrow \Phi_i(a^{(0)} + \langle \mathbf{d}, \mathsf{rot}^0_{q_{\gamma_i},\Phi_k}\rangle)$
 5:     $b^{(i)} \leftarrow \Phi_i(\langle \mathbf{d}, \mathsf{rot}^1_{q_{\gamma_i},\Phi_k}\rangle)$
 6: end for
 7: $(\mathtt{ct}[0], \mathtt{ct}[1]) \leftarrow (0, 0)$
 8: for $j = 0; j < n_2; j++$ do
 9:     $(u^{(0)}, u^{(1)}) \leftarrow (0, 0)$
10:     for $i = 0; i < n_1; i++$ do
11:         $(u^{(0)}, u^{(1)}) \leftarrow (u^{(0)}, u^{(1)}) + (a^{(i)}, b^{(i)}) M^{(n_1 j + i)}_{\mathsf{diag}}$
12:     end for
13:     $u^{(1)} \leftarrow \left\lceil P^{-1} u^{(1)} \right\rfloor$
14:     $\mathbf{d} \leftarrow [[u^{(1)}]_{q_{\gamma_{0\le i<\beta}}}]_{PQ_i}$
15:     $\mathtt{ct}[0] \leftarrow \mathtt{ct}[0] + \Phi_{n_1 j}(u^{(0)} + \langle \mathbf{d}, \mathsf{rot}^0_{q_{\gamma_i},\Phi_k}\rangle)$
16:     $\mathtt{ct}[1] \leftarrow \mathtt{ct}[1] + \Phi_{n_1 j}(\langle \mathbf{d}, \mathsf{rot}^1_{q_{\gamma_i},\Phi_k}\rangle)$
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:     $\mathtt{ct} \leftarrow \mathtt{MatMul}(\mathtt{ct}, ((\frac{1}{2})^{\frac{1}{k}} \mathsf{SF}_i^{-1}))$
3:     $\mathtt{ct} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
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 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:     $\Delta_{i+1} \leftarrow \sqrt{\Delta_i \cdot q_{\ell - \log(d-1) - r + i}}$
 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:     $\delta \leftarrow \delta^2$
12:     $\{\mathtt{ct}, \ell - \log(d-1) - i, \Delta_{r-i}^2 = \Delta_{r-i-1} \cdot q_{\ell - \log(d-1) - i}\} \leftarrow \mathtt{Relinearize}(\mathtt{Tensor}(\mathtt{ct}, \mathtt{ct}))$
13:     $\mathtt{ct} \leftarrow \mathtt{MultConst}(\mathtt{ct}, 2)$
14:     $\mathtt{ct} \leftarrow \mathtt{AddConst}(\mathtt{ct}, -\delta)$
15:     $\{\mathtt{ct}, \ell - \log(d-1) - (i+1), \Delta_{r-i-1}\} \leftarrow \mathtt{Rescale}(\mathtt{ct})$
16: end for
17: return $\{\mathtt{ct}, \ell - \log(d-1) - r, \Delta_0 = \Delta\}$
```

Algorithm 6 EvaluatePolynomial

```
Require: A target scale $\Delta'$, 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: $\mathbf{\Delta'} \leftarrow \mathtt{GetScalingFactors}(p(t), T(\mathtt{ct}), \Delta')$ // Algorithm 7
 6: $b \leftarrow \{\emptyset\}$
 7: for $i \leftarrow 0$ to $2^{m-l} - 1$ do
 8:     $b_i \leftarrow \mathtt{BabyStep}(p_i(t), T, \mathbf{\Delta'}_i)$ // Algorithm 8
 9: end for
10: for $i \leftarrow 0$ to $m - l - 1$ do
11:     for $j \leftarrow 0$ to $(2^{m-l})/(2^{i+1}) - 1$ do
12:         $b_j \leftarrow \mathtt{GiantStep}(b_{2j}, b_{2j+1}, T_{2^{l+i}})$ // Algorithm 9
13:     end for
14: end for
15: return $\mathtt{Rescale}(\mathtt{Relinearize}(b_0))$
```

Algorithm 7 `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 $\Delta'$.
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 GiangStep has valid inputs and the final scale of EvaluatePolynomial is $\Delta'$.
```

Algorithm 8 `BabyStep`

```
Require: A polynomial $p(t) = \sum_{i=0}^{d} c_iT_i(t)$, 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 $\Delta$.
Ensure: $\mathtt{ct}' = \{p(\mathtt{ct}), \ell_{T_d(\mathtt{ct})}, \Delta \cdot q_{\ell_{T_d(\mathtt{ct})}}\}$.
1: $\mathtt{ct}' \leftarrow \lceil c_0 \cdot \Delta \cdot q_{\ell_{T_d}} \rfloor$ // Set to be at level $\ell$
2: for $i \leftarrow 1$ to $d-1$ do
3:     $\mathtt{ct}'' \leftarrow$ 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:     $\mathtt{ct}' \leftarrow$ Add$(\mathtt{ct}', \mathtt{ct}'')$ // $\mathtt{ct}'$ and $a$ both have scaling factor $\Delta \cdot q_{\ell_{T_d}}$
5: return $\mathtt{ct}'$
```

Algorithm 9 `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: $\mathtt{ct}_1 \leftarrow$ Relinearize$(\mathtt{ct}_1)$ //$\Delta_1 q_{\ell_1}$
2: $\mathtt{ct}_1 \leftarrow$ Rescale$(\mathtt{ct}_1)$ //$\Delta_1 q_{\ell_1} \rightarrow \Delta_1$
3: $\mathtt{ct}_1 \leftarrow$ Tensor$(\mathtt{ct}_1, \mathtt{ct}_2)$ //$\Delta_1 \rightarrow \Delta_1\Delta_2$
4: $\mathtt{ct}_1 \leftarrow$ Add$(\mathtt{ct}_1, \mathtt{ct}_0)$ //$\Delta_0 = \Delta_1\Delta_2$
5: return $\mathtt{ct}_1$
```