IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 501–537. DOI:10.46586/tches.v2023.i1.501-537

# FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption

Kamil Kluczniak[1] and Leonard Schild[2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany, kamil.kluczniak@cispa.de

[2] Independent Analyst, Saarbrücken, Germany, schild.leonard@gmail.com

**Abstract.** Computation on ciphertexts of all known fully homomorphic encryption (FHE) schemes induces some noise, which, if too large, will destroy the plaintext. Therefore, the bootstrapping technique that re-encrypts a ciphertext and reduces the noise level remains the only known way of building FHE schemes for arbitrary unbounded computations. The bootstrapping step is also the major efficiency bottleneck in current FHE schemes. A promising direction towards improving concrete efficiency is to exploit the bootstrapping process to perform useful computation while reducing the noise at the same time. We show a bootstrapping algorithm, which embeds a lookup table and evaluates arbitrary functions of the plaintext while reducing the noise. Depending on the choice of parameters, the resulting homomorphic encryption scheme may be either an exact FHE or homomorphic encryption for approximate arithmetic. Since we can evaluate arbitrary functions over the plaintext space, we can use the natural homomorphism of Regev encryption to compute affine functions without bootstrapping almost for free. Consequently, our algorithms are particularly suitable for arithmetic circuits over a finite field with many additions and scalar multiplication gates. We achieve significant speedups when compared to binary circuit-based FHE. For example, we achieve 280-1200x speedups when computing an affine function of size 784 followed by any univariate function when compared to FHE schemes that compute binary circuits. With our bootstrapping algorithm, we can efficiently convert between arithmetic and boolean plaintexts and extend the plaintext space using the Chinese remainder theorem. Furthermore, we can run the computation in an exact and approximate mode where we trade-off the size of the plaintext space with approximation error. We provide a tight error analysis and show several parameter sets for our bootstrapping. Finally, we implement our algorithm and provide extensive tests. We demonstrate our algorithms by evaluating different neural networks in several parameter and accuracy settings.

**Keywords:** Fully Homomorphic Encryption · Bootstrapping

## 1 Introduction

A fully homomorphic encryption scheme provides a way to perform arbitrary computation on encrypted data. The bootstrapping technique, first introduced by Gentry [Gen09], remains thus far the only technique to construct secure fully homomorphic encryption schemes. The reason is that in current homomorphic schemes evaluating encrypted data induces noise, which will eventually "destroy" the plaintext if too high. In practice bootstrapping also remains one of the major efficiency bottlenecks.

The most efficient bootstrapping algorithms to date are the FHEW-style bootstrapping developed by Ducas and Micciancio [DM15], and the TFHE-style bootstrapping by Chillotti et al. [CGGI16, CGGI20]. At a high level, the bootstrapping procedure in these types of bootstrapping algorithms does two things. First, on input, a Regev ciphertext [Reg09]


Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

the bootstrapping algorithm outputs a ciphertext whose error is independent of the error of the bootstrapped ciphertext. Second, it computes a negacyclic function $F$ on the input plaintext. That is $F$ must satisfy $F(x+N) = -F(x) \mod 2 \cdot N$, where $x \in \mathbb{Z}_{2 \cdot N}$. This functionality, together with the linear homomorphism of Regev encryption [Reg09], is enough for FHEW and TFHE to compute arbitrary binary gates. For example, given $\mathsf{Enc}(a \cdot 2N/3)$ and $\mathsf{Enc}(b \cdot 2N/3)$, where $a, b \in \{0,1\}$, we compute the NAND gate by exploiting the negacyclicity. We set $F$ to output $-1$ for $x \in [0, N)$, and from the negacyclicity property we have $F(x) = 1$ for $x \in [N, 2 \cdot N)$. To compute the NAND gate we first compute $\mathsf{Enc}(x) = \mathsf{Enc}(a \cdot 2N/3) + \mathsf{Enc}(b \cdot 2N/3)$ and then $\mathsf{Enc}(c) = \mathsf{Enc}(F(x))$. Note that only for $a = b = 1$, we have $x = 4N/3 > N$ and $c = 1$. For all other valuations of $a$ and $b$ we have $x < N$ and $c = -1$. Now, we can exploit that $c \in \{-1, 1\}$ to choose one of two arbitrary values $y, z \in \mathbb{N}$ by computing $\frac{y \cdot (1+c)}{2} - \frac{z \cdot (c-1)}{2}$. When computing the NAND gate we choose $y = 0$ and $z = 2N/3$. Similarly, we can realize other boolean gates. The crucial observation is that the bootstrapping algorithm relies on the negacyclicity property of the function $F$ to choose an outcome. In particular, the outcome is a binary choice between two values.

Unfortunately, the requirement on the function $F$ stays in the way to efficiently compute useful functions on non-binary plaintexts with only a single bootstrapping operation. For instance, we cannot compute $\frac{1}{1+e^{-x}}$, $\tanh(x)$, $\max(0, x)$, univariate polynomials or other functions that are not negacyclic. Hence, to evaluate such functions on encrypted data, we need to represent them as circuits, encrypt every input digit, and perform a relatively expensive bootstrapping per gate of the circuit. By solving the negacyclicity problem, we can hope for significant efficiency improvements by computing all functions over a larger plaintext space with a single bootstrapping operation. Additionally, we could leverage the natural and extremely efficient linear homomorphism of Regev encryption [Reg09], to efficiently evaluate circuits with a large number of addition and scalar multiplication gates like neural networks.

## 1.1 Contribution

Our main contribution is the design, detailed error analysis, implementation [Sch], and extensive tests of a bootstrapping algorithm that solves the negacyclicity problem. In particular, our bootstrapping algorithm can evaluate all functions over $\mathbb{Z}_t$ for some integer $t$, as it internally embeds a lookup table. We refer to our bootstrapping algorithm as "full domain bootstrapping" as opposed to a version of TFHE [CGGI16, CGGI20, CIM19], sometimes called PBS [CJP21, CLOT21] from "programmable bootstrapping scheme", that can evaluate all functions only over half of the domain $[0, \lfloor t/2 \rfloor)$. We stress that the difference goes far beyond the size of the plaintext space. As we will explain in more detail in Section 1.2, TFHE for non-negacyclic functions cannot exploit the natural linear homomorphism of Regev encryption without any assumptions on the plaintext. Our Full Domain Functional Bootstrapping (FDFB) does not require any assumptions on the plaintext. The ability to compute affine functions "almost for free" gives our FDFB a tremendous advantage in evaluation time for arithmetic circuits with a large number of addition and scalar multiplication gates. We show several parameters sets targeting different bit-precisions and security levels. In particular, we show parameters that allow our FDFB to bootstrap and compute any function $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ with very low error probability, where $t$ is a 7 or 8-bit prime number, but we note that we can bootstrap larger plaintexts when choosing a larger modulus $Q$ and degree $N$ of the ring $\mathcal{R}_Q$. We show that we can take the message modulus higher for approximate arithmetic—for instance, 10 or 11 bits. We also show how to leverage the Chinese remainder theorem to extend the plaintext space to $\mathbb{Z}_t$ for $t$ being a large (e.g., 32-bit or higher) composite integer. In short, we use the fact that the ring $\mathbb{Z}_t$ for $t = \prod_{i=1}^{n} t_i$ where the $t_i$'s are pairwise co-prime is isomorphic with the product ring $\mathbb{Z}_{t_1} \times \cdots \times \mathbb{Z}_{t_n}$. To the best of our knowledge, our paper is the first to utilize the CRT trick in an FHEW/TFHE-style bootstrapping algorithm to extend the plaintext space. Furthermore, the technique was recently included as an important feature in a recent release of a major FHE library [CJL+20]. Finally, we note that our techniques can be directly applied to many other problems like efficient conversion between finite field and binary plaintexts, plaintext equality checks, polynomial evaluation, which efficiency is invariant of the degree of the polynomial, etc.

**Performance.** We implemented our bootstrapping algorithm using the Palisade library [PAL21]. We test the performance of our $\mathsf{FDFB}$ on two use cases. We show the performance for arithmetic circuits modulo $t$. Furthermore, we exemplify the performance of our $\mathsf{FDFB}$ by evaluating neural networks. Roughly speaking, a neural network is a circuit which gates, called neurons, are parameterized by weights $w_0, w_1, \ldots, w_n \in \mathbb{N}$, take as input wires $x_1, \ldots, x_n \in \mathbb{N}$ and output $f(w_0 + \sum_{i=1}^{n} x_i \cdot w_i)$. The function $f$ is typically a non-linear function called the activation function. When homomorphically evaluating an encrypted query to the neural network, we compute the linear combination $w_0 + \sum_{i=1}^{n} x_i \cdot w_i$ by leveraging the natural linear homomorphism of Regev ciphertexts [Reg09]. Finally, we use our $\mathsf{FDFB}$ to compute the activation function $f$. Since our bootstrapping reduces the noise of the ciphertexts along with computing $f$, the time to evaluate the entire network is linear in the number of neurons. In particular, the evaluation time and the parameters for our FHE do not depend on the depth of the neural network. On top of that, our functional bootstrapping algorithm is nicely parallelizable.

We compare our FHE scheme with the state-of-the-art lookup table methods [CGGI16, CGGI20, CGGI17]. To give the best quality of comparison, we rewrite the lookup table algorithms from [CGGI17] in PALISADE [PAL21]. In particular, the arithmetic in $\mathcal{R}_Q$ and other algorithms that our algorithm has in common with [CGGI17] share the same code. We note that the timings in our works and previous works may differ. One reason, among other, is that our implementation is based on the number-theoretic transform as opposed to the fast Fourier transform as in [CGGI16, CGGI20, CGGI17]. Furthermore, we choose new parameters for [CGGI16, CGGI20, CGGI17] that target the same security levels as our bootstrapping algorithm. We report that evaluating neurons with 784 weights is over 280 to 1200 times faster than the lookup table methods, depending on the parameter setting.

## 1.2 Deeper Dive into the Problem and our Solution

We first give a high-level overview of FHEW and TFHE bootstrapping to showcase the problem for using it to compute arbitrary functions. In this section, we keep the exposition rather informal and omit numerous crucial details to highlight the essential ideas underlying our constructions.

**Regev Encryption.** First, let us recall the learning with errors based encryption scheme due to Regev [Reg09]. In the symmetric key setting the encryption algorithm chooses a vector $\mathbf{a} \in \mathbb{Z}_q^n$ and a secret key $\mathbf{s} \in \mathbb{Z}_q^n$, and computes an encryption of a message $\mathsf{m} \in \mathbb{Z}_t$ as $[b, \mathbf{a}^\top]^\top \in \mathbb{Z}_q^{(n+1)\times 1}$, with $b = \mathbf{a}^\top \cdot \mathbf{s} + \tilde{\mathsf{m}} + e \mod q$, where $\tilde{\mathsf{m}} = \frac{q}{t} \cdot \mathsf{m}$ and $e < \frac{q}{2 \cdot t}$ is a "small" error. We assume that $t|q$ for simplicity, but other settings are possible as well. To decrypt, we compute $\left\lfloor \frac{t}{q} \cdot \left([b, \mathbf{a}^\top]^\top \cdot [1, -\mathbf{s}^\top]\right) \right\rceil = \left\lfloor \frac{t}{q} \cdot \left(\frac{q}{t} \cdot \mathsf{m} + e\right) \right\rceil = \mathsf{m}$. The ring version of the encryption scheme is constructed over the cyclotomic ring $\mathcal{R}_Q$ defined by $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. Similarly as for the integer version we choose $\mathfrak{a} \in \mathcal{R}_Q$ and a secret key $\mathfrak{s} \in \mathcal{R}_Q$, and encrypt a message $\mathsf{m} \in \mathcal{R}_t$ as $[\mathfrak{b}, \mathfrak{a}]$ with $\mathfrak{b} = \mathfrak{a} \cdot \mathfrak{s} + \frac{q}{t} \cdot \mathsf{m} + \mathfrak{e}$, where $\mathfrak{e} \in \mathcal{R}_Q$ is a "small" error.

**FHEW/TFHE Bootstrapping.** Now let us proceed to the ideas underlying FHEW [DM15] and TFHE [CGGI16, CGGI20] to bootstrap the LWE ciphertext described above. Both algorithms leverage the structure of the ring $\mathcal{R}_Q$. Recall that $\mathcal{R}_Q$ includes polynomials from $\mathbb{Z}[X]/(X^N + 1)$ that have coefficients in $\mathbb{Z}_Q$. Importantly, in $\mathcal{R}_Q$ the roots of unity form a multiplicative cyclic group $\mathbb{G} = [1, X, \ldots, X^{N-1}, -1, -X, \ldots, -X^{N-1}]$ of order $2 \cdot N$. This is trivial to verify by checking that $X^N \mod (X^N + 1) = -1$. Assume that the LWE modulus is $q = 2 \cdot N$. The concept is to setup a homomorphic accumulator $\mathsf{acc}$ to be an RLWE encryption which initially encrypts the message $\mathsf{rotP} \cdot X^b \in \mathcal{R}_Q$, where $\mathsf{rotP} = \frac{Q}{t} \cdot \left(1 - \sum_{i=2}^{N} X^{i-1}\right) \in R_Q$. Then we multiply $\mathsf{acc}$ with encryptions of $X^{-\mathbf{a}[i]\cdot\mathbf{s}[i]} \in \mathcal{R}_Q$. So that after $n$ iterations the message of the accumulator is set to

$$\begin{aligned} \mathsf{rotP} \cdot X^{b - \sum_{i=1}^{n} \mathbf{a}[i]\cdot\mathbf{s}[i]} &= \mathsf{rotP} \cdot X^{k\cdot q + \tilde{\mathsf{m}} + e} \\ &= \mathsf{rotP} \cdot X^{\tilde{\mathsf{m}} + e \mod 2\cdot N} \in \mathcal{R}_Q. \end{aligned}$$

Note that we set the coefficients of $\mathsf{rotP}$ such that the constant coefficient of the resulting polynomial is $\frac{Q}{t}$ if $\tilde{\mathsf{m}} + e \in [0, N)$, and $-\frac{Q}{t}$ if $\tilde{\mathsf{m}} + e \in [N, 2N)$. We can extract an LWE encryption of the constant term from the rotated accumulator and obtain an LWE ciphertext of the sign of the message without the error term $e$. Clearly, if the most significant bit of $\mathsf{m}$ is set ($\mathsf{m}$ is a negative number) and $e \leq q/2t$, then $\tilde{\mathsf{m}} + e = q/t \cdot \mathsf{m} + e \in [N, 2N)$. Otherwise, $\tilde{\mathsf{m}} + e \in [0, N)$.

We can extend the method to compute other functions simply by setting the coefficient of $\mathsf{rotP}$ to have the desired value after multiplying it by $X^{\tilde{\mathsf{m}}+e}$ in $\mathcal{R}_Q$. To be precise we want to compute $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$, and for our reasoning we use $F : \mathbb{Z}_{2\cdot N} \mapsto \mathbb{Z}_Q$ such that $F(x) = \frac{Q}{t} \cdot f(\lfloor \frac{t}{2\cdot N} \cdot x \rceil)$. Since we work over the multiplicative group $\mathbb{G}$ of order $2 \cdot N$, but $\mathsf{rotP}$ can only have $N$ coefficients, we can only compute functions $F$ such that $F(x + N \mod 2 \cdot N) = -F(x) \mod Q$. In other words, multiplying any $\mathsf{rotP} \in R_Q$ by $X^N$ rotates $\mathsf{rotP}$ by a full cycle negating all its coefficients. In particular, it flips the sign of the constant coefficient. Note that this behavior also affects the function $f$ that we want to compute.

Lu et al. [jLHH+21] claim (see Fig. 2, Theorem 1 in [jLHH+21]) to compute any function with FHEW [DM15]. Unfortunately, they mistakenly assume the roots of unity in $\mathbb{Z}/(X^N + 1)$ form the group $[1, X, \ldots, X^N]$ of order $N$, instead of $\mathbb{G}$ of order $2N$. This issue invalidates Theorem 1 in [jLHH+21] and the correctness of their main contribution, as FHEW is in fact only capable to compute negacyclic functions.

**Problems when Computing Arbitrary Functions.** To compute arbitrary functions $f$ a straightforward solution is to assume that the phase $\tilde{\mathsf{m}} + e$ is always in $[0, N) = [0, q/2)$ which is the case when $\mathsf{m} \in [0, t/2)$. In particular, $\mathsf{m}$ must have its sign fixed. The immediate disadvantage is that we cannot safely compute affine functions on LWE ciphertexts without invoking the bootstrapping algorithm. As an example let us consider $\tilde{\mathsf{m}} = \frac{q}{t}(\mathsf{m}_1 - \mathsf{m}_2)$ and let us ignore for simplicity the error terms. Clearly when $\mathsf{m}_2 > \mathsf{m}_1$ we have $\mathsf{m}_1 - \mathsf{m}_2 \mod t \in [t/2, t)$ and $\tilde{\mathsf{m}} \in [q/2, q)$. In this case, the output of the bootstrapping will be $-F(\tilde{\mathsf{m}} - q/2)$ assuming we set the rotation polynomial to correctly compute $F$ on the interval $[0, q/2)$. To summarize, to use the natural functional bootstrapping correctly, we need to use it as Carpov et al. [CIM19] to compute a lookup table on an array of binary plaintexts that are composed to an integer within the interval $[0, t/2)$. Another solution is to encode the messages in a fixed interval as done by Chillotti et al. [CJP21], and assume the outcome of the affine function does not go outside the interval. Before bootstrapping, the plaintext is shifted to lie in $[0, t/2)$. The interval encoding technique requires to sacrifice at least 2 bits of the plaintext space. We note that previous work including [CJP21] does not give a specification for the encoding method. We learned the method from a private conversation.

**Our Solution: Full Domain Functional Bootstrapping.** Our main contribution is the design of a bootstrapping algorithm that can compute all functions on an encrypted plaintext instead of only negacyclic functions. In particular, we do not need to assume that the plaintext lies in any particular interval. Moreover, we can handle arithmetic in $\mathbb{Z}_t$, where $t$ is a prime or composite integer. Our first observation is that, as shown in

Bourse et al. [BMMP18], we can compute the sign of the message. In particular, given that the message $\mathsf{m}$ is encoded as $\tilde{\mathsf{m}} = \frac{2 \cdot N}{t} \cdot \mathsf{m}$, where $t$ is the plaintext modulus, we have that if $\tilde{\mathsf{m}} \in [0, N)$, then $\mathsf{sgn}(\mathsf{m}) = 1$, and $\tilde{\mathsf{m}} \in [N, 2 \cdot N)$, then $\mathsf{sgn}(\mathsf{m}) = -1$. Note that this function does not return exactly the sign function since it returns 1 for $\mathsf{m} = 0$, but it is enough for our purpose. The idea is to make two bootstrapping operations: the first computes the sign; the second computes the function $F(x)$. Then we multiply the sign and the result of the function. Note that if $x \in [N, 2 \cdot N)$, then $-F(x) \cdot \mathsf{sgn}(x) = F(x)$.

There are two problems with this solution. First of all, we cannot simply multiply two LWE ciphertexts in the leveled mode. While there are methods to do so [Bra12, BV11, BGV12, FV12], in these methods, the noise growth is quadratic and dependent on the size of the plaintexts. Moreover, one of those plaintexts may be large having a significant impact on the parameters. The second problem is that this way, the function $F$ must satisfy $F(x) = F(x + N)$. Thus our primary goal is not satisfied.

Let us first describe how to deal with the second problem. In short, instead of evaluating the sign, we will compute a bit $b$ that indicates the interval in which that plaintext lies. This bit will help us choose the correct output of the evaluated function. That is, we split the function into $F_0$ that computes $F$ correctly for plaintexts in $[0, N)$, and $F_1$ that computes correctly for the remaining plaintexts. To deal with the first problem, we resign of multiplying two ciphertexts at all. Note that a hypothetical multiplication algorithm could now bootstrap the bit $b$ using a bootstrapping that returns a GSW tuple, and in a leveled mode we could then compute $\mathrm{GSW}(1 - b) \cdot \mathsf{LWE}(F_0(x)) + \mathrm{GSW}(b) \cdot \mathsf{LWE}(F_1(x))$. From the properties of the GSW cryptosystem [GSW13], the resulting ciphertexts' noise is at most an additive function of the noises of both ciphertexts. Furthermore, there are implementations for bootstrapping that return GSW ciphertexts [DM15, CGGI17]. We, however, slightly depart from the idea and give a more efficient solution with better error management. Instead of multiplying two ciphertexts, we leverage the fact that the functions $F_0$ and $F_1$ are publicly known, which means that the rotation polynomials for these functions are known as well. Now, based on LWE ciphertexts of a certain form, we will build a new accumulator that will encode one of the given rotation polynomials. Notably, our accumulator builder uses a version of a homomorphic Mux gate. Finally, we will bootstrap using the given accumulator and we obtain $F(x) = F_b(x)$ as desired. We note that this method is roughly twice as fast as creating a GSW ciphertext and multiplying it with LWE ciphertexts.

Now we can compute any function $F : \mathbb{Z}_{2 \cdot N} \mapsto \mathbb{Z}_Q$, and what follows $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$. We note that we can easily generalize the method to also compute functions $f : \mathbb{Z}_t \mapsto \mathbb{Z}_p$ where $t \neq p$. It remains to use a different message scaling in the rotation polynomials. Furthermore, we can compute affine functions over $\mathbb{Z}_t$ without bootstrapping which costs several microseconds per operation (see Table 6).

## 1.3 Related Work

Since Gentry's introduction of the bootstrapping technique [Gen09], the design of fully homomorphic encryption schemes has received extensive study, e.g., [BV11, Bra12, FV12, BGV12, AP13, GSW13, HS15, CH18, HS21]. Brakerski and Vaikuntanathan [BV14] achieved a breakthrough and gave a bootstrapping method that, exploiting the GSW cryptosystem [GSW13] by Gentry, Sahai, and Waters, incurs only a polynomial error growth. These techniques require representing the decryption circuit as a branching program. Alperin-Sheriff and Peikert [AP14] showed a bootstrapping algorithm, where they represent the decryption circuit as an arithmetic circuit and do not rely on Barrington's theorem [Bar89]. Their method exploits the GSW cryptosystem [GSW13] to perform matrix operations on LWE ciphertexts. Hiromasa, Abe, and Okamoto [HAO15] improved the method by designing a version of GSW that natively encrypts matrices. Recently, Genise et al. [GGH+19] showed an encryption scheme that further improves the efficiency of matrix operations. Unfortunately, Lee and Wallet [LW20] showed that the scheme is broken for practical parameters.

Building on the ideas of Alperin-Sheriff and Peikert [AP14], Ducas and Miccancio [DM15] designed a practical bootstrapping algorithm called FHEW. FHEW exploits the ring structure of the RLWE ciphertexts and RGSW noise management for updating a so-called homomorphic accumulator. Chillotti et al. [CGGI16, CGGI20] gave a variant, called TFHE, with numerous optimizations to the FHEW bootstrapping algorithm.

The FHEW and TFHE bootstrapping algorithms constitute state-of-the-art methods to perform practical bootstrapping. Further improvements mostly relied on incorporating packing techniques [CGGI17, MS18], and improved lookup table evaluation [CGGI17, CIM19]. An interesting direction is to exploit the construction of FHEW/TFHE and embed a lookup table directly into the bootstrapping algorithm. To this end, Bonnoron, Ducas, and Fillinger [BDF18] showed a bootstrapping algorithm capable of computing over larger plaintexts, which, however, returns a single bit of output per bootstrapping operation and uses RLWE instantiated over cyclic rings instead of negacyclic. Carpov, Izabachéne, and Mollimard [CIM19] show that we can compute multiple functions on the same input plaintext at the cost of a single TFHE bootstrapping. Notably, their method enjoys amortized time only for functions with a small domain like binary. Guimarães, Borin, and Aranha [GBA21] show several applications of [CIM19]. As mentioned earlier, Lu et al. [jLHH+21] use FHEW to compute negacyclic functions on CKKS [CKKS17] ciphertexts.

**Relation to BFV/BGV [BV11, Bra12, FV12, BGV12] type schemes.** In this setting, we consider FHE with a very low probability of having erroneous outcome of the homomorphic computation. We automatically satisfy CPA$^D$-security [LM20] in contrast to homomorphic encryption for approximate arithmetic. In short, Li and Miccancio [LM20] showed key recovery attacks against some cryptosystems that have a high probability of returning an erroneous plaintext when such plaintext is revealed to a passive adversary. Current bootstrapping algorithms [HS15, CH18, HS21] for BGV/BFV type systems like HeLib [HEl, HS14, HS15, Alb17] or [SEA22] reduce the error of the ciphertext and raise the modulus, without performing any additional computation. To compute in BFV/BGV, we represent the program as an arithmetic circuit. This may be sometimes problematic. For example, *CryptoNets* [DGBL+16], which use the SEAL library to homomorphically evaluate a neural network, cannot easily compute popular activation functions. Therefore, *CryptoNets* uses $x^2$ as an activation function. Furthermore, we need to discretize the trained network by multiplying the weights and biases by a parameter $\delta$ and rounding. Since there is no homomorphic division algorithm, homomorphic computation on discretised data will increase the parameter $\delta$. For example, when computing $\sum_{i=1}^{m} x_i' \cdot w_i'$, where $x_i = \delta \cdot x_i$ and $w_i' = \delta \cdot w_i$ and the $x_i$'s and $w_i$'s are the original floating point inputs and weights, we end up with $\delta^2 \cdot \sum_{i=1}^{m} x_i \cdot w_i$. Note that we ignored the rounding for simplicity. Unfortunately, we cannot rescale the result since there is no natural division algorithm in BFV/BGV. Consequently, *CryptoNets* has to choose enormous parameters that grow exponentially with the depth of the neural network. Our functional bootstrapping algorithm resolves these issues, as we can compute any activation function that is discretized over $\mathbb{Z}_t$. Along the way, we can compute the rescaling, i.e., division by $\delta$ and rounding, together with the activation function within a single bootstrapping operation. This allows us to compute neural networks of unbounded depth without increasing the parameters. On the other hand, we preserve the extremely fast linear operations from BFV/BGV (see Table 5).

**Relation to CKKS/HEAAN [CKKS17] type schemes.** As hinted in the abstract, we can run our bootstrapping algorithm in an approximate mode. In particular, the probability that the error distorts the least significant bits of the message to be bootstrapped grows along with the message space. For example, we choose the plaintext space to be 11-bit, instead of 7-bit as in the "exact" computation setting. However, when using such parameters, we cannot claim that the homomorphic encryption is a "exact" FHE anymore. Consequently, as with all approximate schemes, we cannot formally claim CPA$^D$-security [LM20], but we can apply the same countermeasures [LMSS22] against the Li-Micciancio attack. Nevertheless, this setting allows to use a larger message modulus with the same efficiency and is especially useful when approximate computation is sufficient (e.g., neural networks). Note that without bootstrapping algorithms like ours, current approximate homomorphic encryptions need to approximate a function via a polynomial, discretize its coefficients and evaluate a circuit as deep as the degree of that polynomial. For the approximate homomorphic computation setting, Lu et al. [jLHH$^+$21] proposed to use FHEW [DM15] to compute negacyclic functions only. When using our algorithm, we can compute all functions over the plaintext space without posing any restriction on the plaintexts.

**Applications to Neural Network Inference.** Bourse et al. [BMMP18] applied TFHE [CGGI16, CGGI20] to compute the negacyclic $\mathsf{sgn}(x)$ function as activation for a neural network. Izabachéne, Sirdey, and Zuber [ISZ19] evaluate another neural network (a Hopfield network) using the same method as Bourse et al. [BMMP18]. Chillotti et al. [CJP21] use TFHE to evaluate deep convolution neural networks and report excellent timings for image classification based on the MNIST dataset. Finally, we note that Bourse et al. [BMMP18] left as an open problem to compute general functions with a functional bootstrapping. It is worth noting that oblivious neural network evaluation became a critical use case for fully homomorphic encryption and got much attention and publicity [DGBL$^+$16, CdWM$^+$17, LJLA17, JKLS18, JVC18, KSK$^+$18, KSK$^+$20, BGGJ18, ABSdV19, CDKS19, RSC$^+$19, BGPG20]. These works focus on applying existing homomorphic encryption libraries like SEAL, HeLib, and HEAAN, which we discussed earlier.

**Concurrent and Followup Work.** Concurrently and independently, Chillotti et al. [CLOT21] shows a concept for a full domain functional bootstrapping algorithm that at a very high level is similar to ours. We stress that the methods to achieve FDFB are different. Chillotti et al. [CLOT21] propose to use the BGV cryptosystem instead of GSW as we suggested for a hypothetical solution, to multiply and choose one of two bootstrapped values. In contrast to our work, the authors do not give parameters or an implementation. Nevertheless, we note that due to the use of BGV/BFV style multiplication, the error after bootstrapping grows quadratically and is dependent on the size of the message. In contrast, our algorithm's error is independent of the message. The works from [YXS$^+$21, LMP21] improved and build upon [Klu22] the FDFB technique. Nevertheless, we note that our paper describes additional techniques, like the CRT trick as well as implementation, comparison and analytical details that may be of independent interest.

## 2 Preliminaries

In this section, we recall the algorithms from the literature that we use for our bootstrapping algorithm. Here we give only the interfaces to the algorithm and state their correctness lemmas. However, for reference, we give the specifications and the correctness analysis of all the algorithms in a unified notation in Appendix B and Appendix C respectively.

**Notation.** We denote as $\mathbb{B}$ the set $\{0,1\}$, $\mathbb{R}$ as the set of reals, $\mathbb{Z}$ the set of integers, and $\mathbb{N}$ as the set of natural numbers. Furthermore we denote as $\mathcal{R}_{N,\infty}$ the ring of polynomials $\mathbb{Z}[X]/(X^N+1)$ and as $\mathcal{R}_{N,q} = \mathcal{R}_{N,\infty}/q\mathcal{R}_{N,\infty}$ the ring of polynomials with coefficients in $\mathbb{Z}_q$. We denote vectors with a bold lowercase letter, e.g., $\mathbf{v}$, and matrices with uppercase letters $\mathbf{V}$. We denote a $n$ dimensional column vector as $[f(.,i)]_{i=1}^{n}$, where $f(.,i)$ defines the $i$-th coordinate. For brevity, we will also denote as $[n]$ the vector $[i]_{i=1}^{n}$, and more generally $[n,m]_{i=n}^{m}$ the vector $[n,\ldots,m]^\top$. Finally, let $a = \sum_{i=1}^{N} a_i \cdot X^{i-1}$, be a polynomial with coefficients over $\mathbb{R}$, then we denote $\mathsf{Coefs}(a) = [a_i]_{i=1}^{N} \in \mathbb{R}^{N\times 1}$ the vector of coefficients of the polynomial $a$. For a random variable $a \in \mathbb{Z}$ we denote as $\mathsf{Var}(a)$ the variance of $a$, as $\mathsf{stddev}(a)$ its standard deviation and as $\mathsf{E}(a)$ its expectation. For $\mathfrak{a} \in \mathcal{R}_{N,\infty}$, we define $\mathsf{Var}(\mathfrak{a}) = [\mathsf{Var}(\mathsf{Coefs}(\mathfrak{a}))]_{i=1}^{N}$, $\mathsf{stddev}(\mathfrak{a}) = [\mathsf{stddev}(\mathsf{Coefs}(\mathfrak{a}))]_{i=1}^{N}$ and $\mathsf{E}(\mathfrak{a}) = [\mathsf{E}(\mathsf{Coefs}(\mathfrak{a}))]_{i=1}^{N}$. We denote $||\mathbf{a}||_p = (\sum_{i=1}^{p} |a_i|^p)^{1/p}$ the $p$-norm of a vector $\mathbf{a} \in \mathbb{R}^n$, where $|.|$ denotes the absolute value. For polynomials we compute the $p$-norm by taking its coefficient vector. By $\mathsf{Ham}(\mathbf{a})$ we denote the hamming weight of vector $\mathbf{a}$, i.e., the number of of non-zero coordinates of $\mathbf{a}$. We also define a special symbol $\Delta_{q,t} = \lfloor \frac{q}{t} \rceil$ and a rounding function for an element $\Delta_{q,t} \cdot a \in \mathbb{Z}_q$, as $\lfloor a \rceil_t^q = \lfloor \frac{t}{q} \cdot \Delta_{q,t} \cdot a \rceil$. For ring elements, we take the rounding of the coefficient vector.

Throughout the paper we denote as $q \in \mathbb{N}$ and $Q \in \mathbb{N}$ two moduli. The parameter $n \in \mathbb{N}$ always denotes the dimension of a LWE sample, that we define below. For rings, we always use $N$ to denote the degree of $\mathcal{R}_{N,q}$ or $\mathcal{R}_{N,Q}$. We define $\ell = \lceil \log_{\mathsf{L}} q \rceil$ for some decomposition basis $\mathsf{L} \in \mathbb{N}$. We denote bounds on variances of random variables by $\mathcal{B} \in \mathbb{N}$. Often we mark different decomposition bases $\mathsf{L}_{\mathsf{name}}$ and the corresponding $\ell_{\mathsf{name}}$, or bounds $\mathcal{B}_{\mathsf{name}}$ with some subscript $\mathsf{name}$. Finally, in order not to repeat ourselves and not to overwhelm the reader, we limit ourselves to define certain variables in the definitions of the algorithms, and we refrain to repeat them in the correctness lemmas.

**Learning With Errors.** We recall the learning with errors assumption by Regev [Reg05]. Our description is a generalized version due to Brakerski, Gentry, and Vaikuntanathan [BGV12].

**Definition 1** (Generalized Learning With Errors)**.** Let $\mathcal{X}_{\mathcal{B}} = \mathcal{X}(n)$ be a distribution over $\mathcal{R}_{N,q}$ such that $||\mathsf{Var}(\mathfrak{e})||_\infty \leq \mathcal{B}$ and $\mathsf{E}(\mathfrak{e}) = \mathbf{0}$ for all $\mathfrak{e} \in \mathcal{X}_{\mathcal{B}}$. For $\mathbf{a} \leftarrow_{\$} \mathcal{R}_{N,q}^{n\times 1}$, $\mathfrak{e} \leftarrow_{\$} \mathcal{X}_{\mathcal{B}}$ and $\mathbf{s} \in \mathcal{R}_{N,q}^{n\times 1}$, we define a Generalized Learning With Errors (GLWE) sample of a message $\mathsf{m} \in \mathcal{R}_{N,q}$ with respect to $\mathbf{s}$, as

$$\mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m}) = \begin{bmatrix} \mathfrak{b} = \mathbf{a}^\top \cdot \mathbf{s} + \mathfrak{e} \\ \mathbf{a}^\top \end{bmatrix} + \begin{bmatrix} \mathsf{m} \\ \mathbf{0} \end{bmatrix} \in \mathcal{R}_{N,q}^{(n+1)\times 1}.$$

The $\mathsf{GLWE}_{\mathcal{B},n,N,q}$ problem is to distinguish the following two distributions: In the first distribution, one samples $[\mathfrak{b}_i, \mathbf{a}_i^\top]^\top$ uniformly from $\mathcal{R}_{N,q}^{(n+1)\times 1}$. In the second distribution one first draws $\mathbf{s} \leftarrow_{\$} \mathcal{R}_{N,q}^{n\times 1}$ and then sample $\mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m}) = [\mathfrak{b}_i, \mathbf{a}_i^\top]^\top \in \mathcal{R}_{N,q}^{(n+1)\times 1}$. The $\mathsf{GLWE}_{\mathcal{B},n,N,q}$ assumption is that the $\mathsf{GLWE}_{\mathcal{B},n,N,q}$ problem is hard.

In the definition, we choose the secret key uniformly from $\mathcal{R}_{N,q}$. We note that often secret keys may be chosen from different distributions. When describing our scheme, we will emphasize this when necessary.

We denote an Learning With Errors (LWE) sample as $\mathsf{LWE}_{\mathcal{B},n,q}(\mathbf{s}, \mathsf{m}) = \mathsf{GLWE}_{\mathcal{B},n,1,q}$, which is a special case of a GLWE sample with $N = 1$. Similarly we denote an Ring-Learning with Errors (RLWE) sample as $\mathsf{RLWE}_{\mathcal{B},N,q}(\mathbf{s}, \mathsf{m}) = \mathsf{GLWE}_{\mathcal{B},1,N,q}$ which is the special case of an GLWE sample with $n = 1$. Sometimes we use the notation $\mathbf{c} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s},\ \mathsf{m})$ (resp. $\mathsf{LWE}$ and $\mathsf{RLWE}$) to indicate that a vector $\mathbf{c}$ is a GLWE (resp. LWE and RLWE) sample of the corresponding parameters and inputs. Sometimes we leave the inputs unspecified and substitute them with "." when it is not necessary to refer to them within the scope of a function.

**Definition 2** (Phase and Error for GLWE samples)**.** We define the phase of a sample $\mathbf{c} = \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m})$, as $\mathsf{Phase}(\mathbf{c}) = [1, -\mathbf{s}^\top] \cdot \mathbf{c}$. We define the error of a GLWE sample $\mathsf{Error}(\mathbf{c}) = \mathsf{Phase}(\mathbf{c}) - \mathsf{m}$.

**Lemma 1** (Linear Homomorphism of GLWE samples)**.** *Let* $\mathbf{c} = \mathsf{GLWE}_{\mathcal{B}_{\mathbf{c}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{c}})$ *and* $\mathbf{d} = \mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})$*. If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c} + \mathbf{d}$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m})$*, where* $\mathsf{m} = \mathsf{m}_{\mathbf{c}} + \mathsf{m}_{\mathbf{d}}$*, and* $\mathcal{B} \leq \mathcal{B}_{\mathbf{c}} + \mathcal{B}_{\mathbf{d}}$*. Furthermore, let* $\mathfrak{d} \in \mathcal{R}_{N,\mathsf{L}}$ *where* $\mathsf{L} \in \mathbb{N}$*. If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c} \cdot \mathfrak{d}$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{c}} \cdot \mathfrak{d})$*, where*

- $\mathcal{B} \leq \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$, *when* $\mathfrak{d}$ *is a monomial with* $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$, *and*
- $\mathcal{B} \leq N \cdot \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$, *when* $\mathfrak{d}$ *is a polynomial with* $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$.

**Gentry-Sahai-Waters Encryption.** We recall the cryptosystem from Gentry, Sahai and Waters [GSW13].

**Definition 3** (Gadget Matrix)**.** We call the column vector $\mathbf{g}_{\ell,\mathsf{L}} = [\mathsf{L}^{i-1}]_{i=1}^{\ell} \in \mathbb{N}^{\ell \times 1}$ powers-of-$\mathsf{L}$ vector. For some $k \in \mathbb{N}$ we define the gadget matrix as $\mathbf{G}_{\ell,\mathsf{L},k} = \mathbf{I}_k \otimes \mathbf{g}_{\ell,\mathsf{L}} \in \mathbb{N}^{k\ell \times k}$, where $\otimes$ denotes the Kronecker product.

**Definition 4** (Decomposition)**.** We define the $\mathsf{G}_{\mathsf{L},q}^{-1}$ function to take as input an element $\mathfrak{a} \in \mathcal{R}_{N,q}$ and return a row vector $\mathbf{a} = [\mathfrak{a}_1, \ldots, \mathfrak{a}_\ell] \in \mathcal{R}_{\mathsf{L}}^{1\times\ell}$ such that $\mathfrak{a} = \mathbf{a} \cdot \mathbf{g}_{\ell,\mathsf{L}} = \sum_{i=1}^{\ell} \mathfrak{a}_i \cdot \mathsf{L}^{i-1}$. Furthermore we generalize the function to matrices where $\mathsf{G}_{\mathsf{L},q}^{-1}$ is applied to every entry of the input matrix. Specifically, on input a matrix $\mathbf{M} \in \mathcal{R}_{N,q}^{m\times k}$, $\mathsf{G}_{\mathsf{L},q}^{-1}$ outputs a matrix $\mathbf{D} \in \mathcal{R}_{\mathsf{L}}^{m\times k\ell}$.

**Definition 5** (Generalized-GSW Sample)**.** Let $\mathbf{s} \in \mathcal{R}_{N,q}^{n}$ and $\mathsf{m} \in \mathcal{R}_{N,q}$. We define Generalized-GSW (GGSW) samples as $\mathsf{GGSW}_{\mathcal{B},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}) = \mathbf{A} + \mathsf{m} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)} \in \mathcal{R}_{N,q}^{(n+1)\ell\times(n+1)}$, where $\mathbf{A}[i,*] = \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, 0)^\top$ for all $i \in [(n+1)\ell]$. In other words, all rows of $\mathbf{A}$ consist of (transposed) GLWE samples of zero.

Similarly, as with LWE and RLWE, we denote an GSW sample as $\mathsf{GSW}_{\mathcal{B},n,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}) = \mathsf{GGSW}_{\mathcal{B},n,1,q,\mathsf{L}}(\mathbf{s}, \mathsf{m})$, which is a special case of a GGSW sample with $N = 1$. Similarly we will denote an Ring-GSW (RGSW) sample as $\mathsf{RGSW}_{\mathcal{B},N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}) = \mathsf{GGSW}_{\mathcal{B},1,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m})$, which is the special case of an GGSW sample with $n = 1$. Will use the notation $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m})$ (resp. GSW and RGSW) to indicate that a matrix $\mathbf{C}$ is a GGSW (resp. GSW and RGSW) sample with the corresponding parameters and inputs.

**Definition 6** (Phase and Error for GGSW samples)**.** We define the phase of a sample $\mathbf{C} = \mathsf{GGSW}_{\mathcal{B},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}) = \mathbf{A} + \mathsf{m} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)}$ as $\mathsf{Phase}(\mathbf{C}) = \mathbf{C} \cdot [1, -\mathbf{s}^\top]^\top$. We define the error of a GLWE sample as $\mathsf{Error}(\mathbf{C}) = \mathsf{Phase}(\mathbf{C}) - \mathsf{m} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)} \cdot [1, -\mathbf{s}^\top]^\top \in \mathcal{R}_{N,q}^{(n+1)\ell\times 1}$.

**Lemma 2** (Linear Homomorphism of GGSW samples)**.** *Let* $\mathbf{C} = \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$ *and* $\mathbf{D} = \mathsf{GGSW}_{\mathcal{B}_{\mathbf{D}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{D}})$. *If* $\mathbf{C}_{\mathsf{out}} \leftarrow \mathbf{C} + \mathbf{D}$, *then* $\mathbf{C}_{\mathsf{out}} \in \mathsf{GGSW}_{\mathcal{B},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m})$, *where* $\mathsf{m} = \mathsf{m}_{\mathbf{C}} + \mathsf{m}_{\mathbf{D}}$, *and* $\mathcal{B} \leq \mathcal{B}_{\mathbf{C}} + \mathcal{B}_{\mathbf{D}}$. *Furthermore, let* $\mathfrak{d} \in \mathcal{R}_{\mathsf{L}_{\mathfrak{d}}}$, *where* $\mathsf{L}_{\mathfrak{d}} \in \mathbb{N}$. *If* $\mathbf{C}_{\mathsf{out}} \leftarrow \mathbf{C} \cdot \mathfrak{d}$, *then* $\mathbf{C}_{\mathsf{out}} \in \mathsf{GGSW}_{\mathcal{B},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}} \cdot \mathfrak{d})$ *where*

- $\mathcal{B} \leq \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$, *when* $\mathfrak{d}$ *is a monomial with* $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$, *and*
- $\mathcal{B} \leq N \cdot \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$, *when* $\mathfrak{d}$ *is a polynomial with* $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$.

The external product multiplies an RGSW sample with an RLWE sample resulting in an RLWE sample of the product of their messages.

**Definition 7** (External Product)**.** The external product $\mathsf{extProd}$ given as input $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$ and $\mathbf{d} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})$, outputs $\mathsf{extProd}(\mathbf{C}, \mathbf{d}) = \mathsf{G}_{\mathsf{L},q}^{-1}(\mathbf{d}^\top) \cdot \mathbf{C}$.

**Lemma 3** (Correctness of the External Product)**.** *Let* $\mathbf{C} = \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$, *where* $\mathsf{m}_{\mathbf{C}}$ *consists of a single coefficient, and* $\mathbf{d} = \mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})$. *If* $\mathbf{c} \leftarrow \mathsf{extProd}(\mathbf{C}, \mathbf{d})$, *then* $\mathbf{c}^\top \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m})$, *where* $\mathsf{m} = \mathsf{m}_{\mathbf{C}} \cdot \mathsf{m}_{\mathbf{d}}$ *and*

- $\mathcal{B} \leq N \cdot (n+1) \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{C}} + N \cdot \mathcal{B}_{\mathsf{m}_{\mathbf{C}}}^2 \cdot \mathcal{B}_{\mathbf{d}}$ *in general, and*
- $\mathcal{B} \leq N \cdot (n+1) \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{C}} + \mathcal{B}_{\mathsf{m}_{\mathbf{C}}}^2 \cdot \mathcal{B}_{\mathbf{d}}$ *when* $\mathsf{m}_{\mathbf{C}}$ *is a monomial.*

*with* $\mathcal{B}_{\mathsf{m}_{\mathbf{C}}} \leq ||\mathsf{m}_{\mathbf{C}}||_\infty$.

**Mux Gate.** Informally, the $\mathsf{Mux}$ function takes as input a control GGSW sample and two GLWE samples. The gate outputs one of the input GLWE samples depending on the bit encrypted in the GGSW sample.

**Definition 8** (Mux Gate)**.** The $\mathsf{Mux}$ function is defined as follows.

- $\mathsf{Mux}(\mathbf{C}, \mathbf{g}, \mathbf{h})$: The algorithms takes as input $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s},\ \mathsf{m}_{\mathbf{C}})$, where $\mathsf{m}_{\mathbf{C}} \in \mathbb{B}$, $\mathbf{g} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{g}},n,N,q}(\mathbf{s},\ .)$ and $\mathbf{h} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{h}},n,N,q}(\mathbf{s},\ .)$, and returns $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s},\ .)$.

**Lemma 4** (Correctness of the $\mathsf{Mux}$ gate)**.** *Let* $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$*, where* $\mathsf{m}_{\mathbf{C}} \in \mathbb{B}$ *. Let* $\mathbf{g} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{g}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{g}})$ *and* $\mathbf{h} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{h}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{h}})$*.*

*If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{Mux}(\mathbf{C}, \mathbf{g}, \mathbf{h})$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathsf{out}})$*, where* $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{h}}$ *for* $\mathsf{m}_{\mathbf{C}} = 0$ *and* $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{g}}$ *for* $\mathsf{m}_{\mathbf{C}} = 1$*, and* $\mathcal{B} \leq \frac{1}{3} \cdot (n+1) \cdot N \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{C}} + \max(\mathcal{B}_{\mathbf{g}}, \mathcal{B}_{\mathbf{h}})$*.*

**Modulus Switching.** The modulus switching technique, developed by Brakerski and Vaikuntanathan [BV11], allows an evaluator to change the modulus of a given ciphertext without the knowledge of the secret key.

**Definition 9** (Modulus Switching for LWE Samples)**.** We define modulus switching from $\mathbb{Z}_Q$ to $\mathbb{Z}_q$ by the following algorithm.

- $\mathsf{ModSwitch}(\mathbf{c}, Q, q)$: On input a LWE sample $\mathbf{c} = [b, \mathbf{a}^\top]^\top \in \mathsf{LWE}_{\mathcal{B},n,Q}(\mathbf{s}, .)$, and moduli $Q$ and $q$, the function outputs $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B},n,q}(\mathbf{s}, .)$.

**Lemma 5** (Correctness of Modulus Switching)**.** *Let* $\mathbf{c} = \mathsf{LWE}_{\mathcal{B}_{\mathbf{c}},n,Q}(\mathbf{s}, \mathsf{m} \cdot (\frac{Q}{t}))$ *where* $\mathbf{s} \in \mathbb{Z}_q^n$ *and* $Q = 0 \mod t$*. If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}, Q, q)$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B},n,q}(\mathbf{s}, \mathsf{m} \cdot (\frac{q}{t}))$*, where*

$$\mathcal{B} \leq \frac{q^2}{Q^2} \cdot \mathcal{B}_{\mathbf{c}} + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty.$$

**Sample Extraction.** Informally, sample extraction allows extracting from an RLWE sample an LWE sample of a single coefficient without increasing the error rate. Here we give a generalized version of the extraction algorithm, which can extract an LWE sample of any coefficient of the RLWE sample.

**Definition 10** (Sample Extraction)**.** Sample extraction consists of algorithms $\mathsf{KeyExt}$ and $\mathsf{SampleExt}$.

- $\mathsf{KeyExt}(\mathfrak{s})$: Takes as input a key $\mathfrak{s} \in \mathcal{R}_{N,q}$, and outputs $\mathsf{Coefs}(\mathfrak{s})$.
- $\mathsf{SampleExt}(\mathsf{ct}, k)$: Takes as input $\mathsf{ct} = [\mathfrak{b}, \mathfrak{a}]^\top \in \mathsf{RLWE}_{\mathcal{B},N,q}(\mathfrak{s}, .)$ and an index $k \in [N]$. The function outputs $\mathbf{c} \in \mathsf{LWE}_{\mathcal{B},N,q}(\mathbf{s}, .)$.

**Lemma 6** (Correctness of Sample Extraction)**.** *Let* $\mathsf{ct} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{ct}},N,q}(\mathfrak{s}, \mathsf{m})$*, where* $\mathfrak{s}, \mathsf{m} \in \mathcal{R}_{N,q}$*. Denote* $\mathsf{m} = \sum_{i=1}^{N} \mathsf{m}_i \cdot X^{i-1}$*, where* $\mathsf{m}_i \in \mathbb{Z}_q$*. If* $\mathbf{s} \leftarrow \mathsf{KeyExt}(\mathfrak{s})$ *and* $\mathbf{c} \leftarrow \mathsf{SampleExt}(\mathsf{ct}, k)$*, for some* $k \in [N]$*, then* $\mathbf{c} \in \mathsf{LWE}_{\mathcal{B},N,q}(\mathbf{s}, \mathsf{m}_k)$*, where* $\mathcal{B} \leq \mathcal{B}_{\mathsf{ct}}$*.*

**Key Switching.** Key-switching is an important technique to build scalable homomorphic encryption schemes. In short, having a key switching key, the evaluator can map a given LWE sample to an LWE sample of a different key.

**Definition 11** (Key Switching LWE to GLWE)**.** We recall key generation $\mathsf{KeySwitchSetup}$ and LWE to GLWE switching $\mathsf{KeySwitch}$ as follows.

- $\mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, n, n', N, q, \mathbf{s}, \mathbf{s}', \mathsf{L}_{\mathsf{ksK}})$: Takes as input bound $\mathcal{B}_{\mathsf{ksK}} \in \mathbb{N}$, dimensions $n, n' \in \mathbb{N}$, a degree $N \in \mathbb{N}$ and a modulus $q \in \mathbb{N}$, vectors $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{s}' \in \mathcal{R}_{N,q}^{n'}$ and a basis $\mathsf{L}_{\mathsf{ksK}} \in \mathbb{N}$. The algorithm outputs $\mathsf{ksK} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', .)^{n \times \ell_{\mathsf{ksK}}}$.

- $\mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK})$: Takes as input $\mathbf{c} = [b, \mathbf{a}^\top]^\top \in \mathsf{LWE}_{\mathcal{B}_\mathbf{c},n,q}(\mathbf{s}, .)$ and a key switching key $\mathsf{ksK} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', .)^{n \times \ell_{\mathsf{ksK}}}$. The algorithm outputs $c_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n',N,q}(\mathbf{s}', .)$.

**Lemma 7** (Correctness of Key Switching)**.** *If* $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, n, n', N, q, \mathbf{s}, \mathbf{s}', \mathsf{L}_{\mathsf{ksK}})$ *and* $c_{\mathsf{out}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK})$ *then* $c_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n',N,q}(\mathbf{s}', \mathsf{m})$*, where*

$$\mathcal{B} \leq \mathcal{B}_\mathbf{c} + n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}$$

*and* $\ell_{\mathsf{ksK}} = \lceil \log_{\mathsf{L}_{\mathsf{ksK}}} q \rceil$*.*

**TFHE Blind Rotation and Bootstrapping.** Below we recall the blind rotation introduced by Chillotti et al. [CGGI16]. For both blind rotation and bootstrapping we use a decomposition vector $\mathbf{u} \in \mathbb{Z}^u$ for which the following holds. For all $y \in \mathcal{S}$ with $\mathcal{S} \subseteq \mathbb{Z}_{2 \cdot N}$ there exists $\mathbf{x} \in \mathbb{B}^u$ such that $y = \sum_{i=1}^{u} \mathbf{x}[i] \cdot \mathbf{u}[i] \mod 2 \cdot N$. For example for $\mathcal{S} = \{0, 1\}$ we have $\mathbf{u} = \{1\}$, and for $\mathcal{S} = \{-1, 0, 1\}$ we have $\mathbf{u} = \{-1, 1\}$.

**Definition 12** (TFHE Blind Rotation)**.** Blind rotation consists of a key generation algorithm $\mathsf{BRKeyGen}$, and blind rotation algorithm $\mathsf{BlindRotate}$.

- $\mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$: Takes as input a dimension $n$, a secret key $\mathbf{s} \in \mathbb{Z}_t^{n \times 1}$ and a vector $\mathbf{u} \in \mathbb{Z}^u$, a bound $\mathcal{B}_{\mathsf{brK}}$, a degree $N$ and a modulus $Q$ defining the ring $\mathcal{R}_{N,Q}$, a basis $\mathsf{L}_{\mathsf{RGSW}} \in \mathbb{N}$, and a secret key $\mathfrak{s} \in \mathcal{R}_{N,Q}$. The algorithm outputs a blind rotation key $\mathsf{brK}$.
- $\mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}, \mathsf{ct}, \mathbf{u})$: This algorithm takes as input a blind rotation key $\mathsf{brK} = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{RGSW}}}(\mathfrak{s}, .)^{n \times u}$, an accumulator $\mathsf{acc} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{acc}},N,Q}(\mathfrak{s}, .)$, a ciphertext $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,2 \cdot N}(\mathbf{s}, .)$, where $\mathbf{s} \in \mathbb{Z}_t^n$ with $t \leq 2 \cdot N$, and a vector $\mathbf{u} \in \mathbb{Z}^u$. The algorithm outputs $\mathsf{acc}_{\mathsf{out}}$.

**Lemma 8** (Correctness of TFHE Blind-Rotation)**.** *Let* $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,2 \cdot N}(\mathbf{s}, .)$ *be a LWE sample and denote* $\mathsf{ct} = [b, \mathbf{a}^\top]^\top \in \mathbb{Z}_{2 \cdot N}^{(n+1) \times 1}$*. Let* $\mathsf{acc} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{acc}},N,Q}(\mathfrak{s}, \mathsf{m}_{\mathsf{acc}})$*. If* $\mathsf{brK} \leftarrow \mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$ *and* $\mathsf{acc}_{\mathsf{out}} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}, \mathsf{ct}, \mathbf{u})$*, then* $\mathsf{acc}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,Q}(\mathfrak{s}, \mathsf{m}_{\mathsf{acc}}^{\mathsf{out}})$*, where* $\mathsf{m}_{\mathsf{acc}}^{\mathsf{out}} = \mathsf{m}_{\mathsf{acc}} \cdot X^{b - \mathbf{a}^\top \cdot \mathbf{s} \mod 2 \cdot N} \in \mathcal{R}_{N,Q}$ *and*

$$\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_{\mathsf{acc}} + 2 \cdot n \cdot u \cdot N \cdot \ell_{\mathsf{RGSW}} \cdot \mathsf{L}_{\mathsf{RGSW}}^2 \cdot \mathcal{B}_{\mathsf{brK}}$$

**Definition 13** (Bootstrapping)**.** Bootstrapping is as follows.

- $\mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}, \mathsf{ksK})$: This algorithm takes as input a blind rotation key $\mathsf{brK} = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{brK}}}(\mathfrak{s}, .)^{n \times u}$, a vector $\mathbf{u} \in \mathbb{Z}^u$, a LWE sample $\mathsf{ct} = \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,q}(\mathbf{s}, .) = [b, \mathbf{a}^\top]^\top \in \mathbb{Z}_q^{(n+1) \times 1}$, a polynomial $\mathsf{rotP} \in \mathcal{R}_{N,Q}$, and a LWE to LWE key switching key $\mathsf{ksK}$. The algorithm outputs $\mathsf{ct}_{\mathsf{out}}$.

**Theorem 1** (Correctness of TFHE Bootstrapping)**.** *Let* $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m})$*. Let* $\mathsf{brK} \leftarrow \mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$*,* $\mathbf{s}_F \leftarrow \mathsf{KeyExt}(\mathfrak{s})$*, and* $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, N, n, q, \mathbf{s}_F, \mathbf{s}, \mathsf{L}_{\mathsf{ksK}})$*. Let* $\mathsf{ct}_N \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q, 2 \cdot N) \in \mathsf{LWE}_{\mathcal{B}_N,n,2 \cdot N}(\mathbf{s}, \Delta_{2 \cdot N,t} \cdot \mathsf{m})$*. Let* $\tilde{\mathsf{m}} \leftarrow \mathsf{Phase}(\mathsf{ct}_N)$*,* $\mathsf{m} = \lfloor \tilde{\mathsf{m}} \rceil_t^{2 \cdot N}$ *and* $\mathsf{rotP} \in \mathcal{R}_{N,Q}$ *be such that if* $\tilde{\mathsf{m}} \in [0, N)$*, then* $\mathsf{Coefs}(X^{\tilde{\mathsf{m}}} \cdot \mathsf{rotP})[1] = \Delta_{Q,t} \cdot F(\mathsf{m})$*, where* $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$*. If* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}, \mathsf{ksK})$ *and* $\mathsf{m} \leq \frac{t}{2} - 1$*, then* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathbf{s}, \mathsf{m}_{\mathsf{out}})$*, where* $\mathsf{m}_{\mathsf{out}} = \Delta_{Q,t} \cdot F(\mathsf{m})$ *and*

$$\mathcal{B}_{\mathsf{out}} \leq \Big(\frac{q^2}{Q^2}\Big) \cdot (\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}) + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty,$$

*where* $\mathcal{B}_{\mathsf{BR}} \leq 2 \cdot n \cdot u \cdot N \cdot \ell_{\mathsf{RGSW}} \cdot \mathsf{L}_{\mathsf{RGSW}}^2 \cdot \mathcal{B}_{\mathsf{brK}}$ *and* $\mathcal{B}_{KS} \leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}$*. If* $Q = q$*, then* $\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}$*. Finally we have,*

$$\mathcal{B}_N \leq \Big(\frac{2 \cdot N}{q}\Big)^2 \cdot \mathcal{B}_{\mathsf{ct}} + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty$$

**Table 1: List of procedures.** The table contains an informal summary of the procedures from Section 2.

| Name | Description |
|---|---|
| $\mathsf{extProd}$ | On input $\mathbf{C} = \mathsf{GGSW}(\mathbf{s}, m_{\mathbf{C}})$ and $\mathbf{d} = \mathsf{GLWE}(\mathbf{s}, m_{\mathbf{d}})$, $\mathsf{extProd}$ computes the product $\mathbf{c}^\top = \mathsf{GLWE}(\mathbf{s}, m)$ with $m = m_{\mathbf{C}} \cdot m_{\mathbf{d}}$ |
| $\mathsf{Mux}$ | Given $\mathbf{C} = \mathsf{GGSW}(\mathbf{s}, m_{\mathbf{C}})$, $\mathbf{g} = \mathsf{GLWE}(\mathbf{s}, m_{\mathbf{g}})$ and $\mathbf{h} = \mathsf{GLWE}(\mathbf{s}, m_{\mathbf{h}})$ with $m_{\mathbf{C}} \in \mathbb{B}$, $\mathsf{Mux}$ outputs $\mathbf{c} = \mathsf{GLWE}(\mathbf{s}, m)$ with $m = m_{\mathbf{g}}$ if $m_{\mathbf{C}} = 1$ and $m = m_{\mathbf{h}}$ otherwise. |
| $\mathsf{ModSwitch}$ | Given a LWE sample $\mathbf{c}$ under source modulus $Q$ and a target modulus $q$, $\mathsf{ModSwitch}$ returns a LWE sample $\mathbf{c}'$ under modulus $q$ encoding the same message as $\mathbf{c}$, w.r.t to a message space $t$. |
| $\mathsf{SampleExt}$ | On input a $\mathsf{RLWE}$ sample $\mathsf{ct}$ and index $k \in \mathbb{N}$, $\mathsf{SampleExt}$ outputs a $\mathsf{LWE}$ sample $\mathsf{ct}'$ encoding the $k$-th coefficient of the polynomial encrypted in $\mathsf{ct}$. |
| $\mathsf{KeySwitchSetup}$ | On input a set of parameters, and two $\mathsf{LWE}$ secret keys $\mathbf{s}, \mathbf{s}'$, $\mathsf{KeySwitchSetup}$ generates a key-switching key $\mathsf{ksK}$. |
| $\mathsf{KeySwitch}$ | On input a key-switching key $\mathsf{ksK}$ for the LWE keys $\mathbf{s}, \mathbf{s}'$ and a $\mathsf{LWE}$ sample $\mathsf{ct}$ under key $\mathbf{s}$, $\mathsf{KeySwitch}$ outputs a $\mathsf{LWE}$ sample $\mathsf{ct}'$ under key $\mathbf{s}'$ encoding the same message as $\mathsf{ct}$, w.r.t to a message space $t$. |
| $\mathsf{BRKeyGen}$ | On input a set of parameters, $\mathsf{LWE}$ key $\mathbf{s}$ and $\mathsf{RLWE}$ key $\mathfrak{s}$, $\mathsf{BRKeyGen}$ generates a blind rotation key. |
| $\mathsf{BlindRotate}$ | Given a blind rotation key, a $\mathsf{LWE}$ sample $\mathsf{ct}$ under modulus $2N$ and an accumulator $\mathsf{acc} = \mathsf{RLWE}(\mathfrak{s}, \mathbf{m}_{\mathsf{acc}})$, $\mathsf{BlindRotate}$ returns a sample $\mathsf{acc}_{\mathrm{out}} = \mathsf{RLWE}(\mathfrak{s}, \mathbf{m}_{\mathsf{out}})$ with $\mathbf{m}_{\mathsf{out}} = \mathbf{m}_{\mathsf{acc}} \cdot X^{\mathsf{Phase}(\mathsf{ct})}$ |
| $\mathsf{Bootstrap}$ | Given a blind rotation key, a key switching key, a sample $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, \Delta_{q,t} \cdot m)$ and a polynomial $\mathsf{rotP}$ correctly encoding a function $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$, $\mathsf{Bootstrap}$ returns $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}(\mathbf{s}, m_{\mathsf{out}})$ such that $m_{\mathsf{out}} = \Delta_{q,t} \cdot F(m)$ if $m < \frac{t}{2}$ and $m_{\mathsf{out}} = -\Delta_{q,t} \cdot F(m)$ otherwise |

*if $N \neq 2 \cdot q$, and $\mathcal{B}_N = \mathcal{B}_{\mathsf{ct}}$ otherwise, where $\mathcal{B}_{\mathsf{ct}} = \max(\mathcal{B}_{\mathsf{fresh}}, \mathcal{B}_{\mathsf{out}})$ and $\mathcal{B}_{\mathsf{fresh}}$ is the bound in the error variance of a fresh ciphertext.*

# 3 Our Functional Bootstrapping Technique

In this section, we show our functional bootstrapping algorithm. First, however, we show two sub-procedures that help us construct the bootstrapping procedure.

## 3.1 Public Mux Gate and Building a Homomorphic Accumulator

We describe a public version of the homomorphic Mux gate which aims to choose one of two given polynomial plaintexts based on an encrypted bit. Furthermore, we show how to use the public Mux gate to build an accumulator for the functional bootstrap. Roughly speaking, the accumulator builder gets an encrypted bit and based on that bit chooses one of two different rotation polynomials. The difference between the accumulator builder and the public mux is that our accumulator builder needs to switch the encrypted bit from LWE samples to RLWE samples. The algorithms are depicted in Figure 1 and Figure 2. Below we state the correctness theorems for both algorithms, but we devise the proofs to Appendix A.

**Lemma 9** (Correctness of Public Mux)**.** *Let $[\mathbf{c}_i]_{i=1}^{\ell}$ be a vector where $\mathbf{c}_i \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{c}},n,N,q}(\mathbf{s},$*

$\mathsf{PubMux}([\mathbf{c}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1)$

**Input:** Takes as input a vector $[\mathbf{c}_i]_{i=1}^{\ell}$ where $\mathbf{c}_i \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m} \cdot \mathsf{L}^{i-1})$ where $\mathsf{m} \in \{0,1\}$ and $\ell = \lceil \log_{\mathsf{L}} q \rceil$, and $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathcal{R}_{N,q}$.
Note that when $N = 1$, we have $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathbb{Z}_q$.
**Output:** A ciphertext $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{out}},n,N,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$.

1 : Set $\mathfrak{p} = \mathfrak{p}_1 - \mathfrak{p}_0$.
2 : Set $\mathbf{p} = \mathsf{G}_{\mathsf{L},q}^{-1}(\mathfrak{p})$.
3 : Compute $\mathbf{c}_{\mathsf{out}} \leftarrow [\Delta_{q,t} \cdot \mathfrak{p}_0, \quad \mathbf{0}]^\top + \sum_{i=1}^{\ell} \mathbf{p}[i] \cdot \mathbf{c}_i$.
4 : Return $\mathbf{c}_{\mathsf{out}}$.

**Figure 1:** Public Mux algorithm

$\mathsf{BuildAcc}([\mathsf{acc}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1, \mathsf{ksK})$

**Input:** Takes as input a vector $[\mathsf{acc}_i]_{i=1}^{\ell}$ where $\mathsf{acc}_i \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{acc}},n,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m} \cdot \mathsf{L}^{i-1})$, where $\mathsf{m} \in \{0,1\}$ and $\ell = \lceil \log_{\mathsf{L}} q \rceil$, $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathcal{R}_{N,q}$ and a key switching key $\mathsf{ksK}$.
**Output:** A ciphertext $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathfrak{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$.

1 : For all $i \in [\ell]$ compute $\mathsf{acc}_{R,i} \leftarrow \mathsf{KeySwitch}(\mathsf{acc}_i, \mathsf{ksK})$.
2 : Compute $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{PubMux}([\mathsf{acc}_{R,i}]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1)$.
3 : Output $\mathbf{c}_{\mathsf{out}}$.

**Figure 2:** Accumulator builder algorithm.

$\mathsf{m} \cdot \Delta_{q,t} \cdot \mathsf{L}^{i-1})$ *where* $n, N, q \in \mathbb{N}$, $\mathsf{m} \in \{0,1\}$, $\ell = \lceil \log_{\mathsf{L}} q \rceil$. *Let* $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathcal{R}_{N,q}$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{PubMux}([\mathbf{c}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1)$, *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{out}},n,N,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$, *where* $\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{c}}$. *If* $\mathfrak{p}_0, \mathfrak{p}_1$ *are monomials then* $N$ *disappears from the above inequality.*

**Lemma 10** (Correctness of the Accumulator Builder)**.** *Let* $[\mathsf{acc}_i]_{i=1}^{\ell}$ *be a vector of LWE samples* $\mathsf{acc}_i \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{acc}},n,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m} \cdot \mathsf{L}^{i-1})$, *where* $\mathsf{m} \in \{0,1\}$, $n, q \in \mathbb{N}$, $\mathcal{B}_{\mathsf{acc}} \leq q$, $\mathbf{s} \in \mathbb{Z}_q^{n \times 1}$, *and* $\ell = \lceil \log_{\mathsf{L}} q \rceil$. *Let* $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, n, 1, N, q, \mathbf{s}, \mathfrak{s}, \mathsf{L}_{\mathsf{ksK}})$, *where* $N \in \mathbb{N}$, $\mathfrak{s} \in \mathcal{R}_{N,q}$ *and* $\mathsf{L}_{\mathsf{ksK}} \leq q$. *Finally let* $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathcal{R}_{N,q}$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{BuildAcc}([\mathsf{acc}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1, \mathsf{ksK})$, *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathfrak{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$, *where*

$$\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot \left(\mathcal{B}_{\mathsf{acc}} + n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}\right).$$

*If* $\mathfrak{p}_0, \mathfrak{p}_1$ *are monomials, then the degree* $N$ *disappears from the above inequality.*

## 3.2 Construction of the Rotation Polynomial

In Figure 3 we describe how to build the polynomials for our bootstrapping to compute arbitrary functions, and by Theorem 2, we show the correctness of our methodology. We defer the proof to Appendix A. We want to evaluate $\mathsf{BootsMap} : \mathbb{Z}_t \mapsto \mathbb{Z}_Q$. Informally, we partition the rotation polynomials into chunks where each chunk represents a value in the domain of $\mathsf{BootsMap}$. The chunk size is determined by the error bound of the bootstrapped LWE sample. Note that $\mathsf{BootsMap}$ already assumes that the domain is scaled to fit the ciphertext modulus. That is, if we want to compute $f : \mathbb{Z}_t \mapsto \mathbb{Z}_{t'}$ for some $t' \in \mathbb{N}$, then we set the lookup table $\mathsf{BootsMap}[x] = \Delta_{Q,t'} \cdot f(x)$ for all $x \in \mathbb{Z}_t$.

$\mathsf{SetupPolynomial}(\mathsf{BootsMap}, N)$

**Input:** A lookup table $\mathsf{BootsMap} : \mathbb{Z}_t \to \mathbb{Z}_Q$that maps elements **in** $\mathbb{Z}_t$ **to** elements **in** $\mathbb{Z}_Q$, and the degree that defines the ring $\mathcal{R}_{N,Q}$.

**Output:** Rotation polynomials $\mathsf{rotP}_0, \mathsf{rotP}_1 \in \mathcal{R}_{N,Q}$.

For $\mathsf{m} \in [0, t-1]$ and **for** $e \in \left[ -\lceil \frac{N}{t} \rceil, \lceil \frac{N}{t} \rceil \right]$ do
  Set $y \leftarrow \lfloor \frac{2 \cdot N}{t} \rceil \cdot \mathsf{m} + e \pmod{2N}$.
  If $y = 0$,
    set $\mathbf{p}_0[1] \leftarrow \mathsf{BootsMap}[\mathsf{m}+1]$.
  If $y \in [1, N-1]$,
    set $\mathbf{p}_0[N-y+1] \leftarrow -\mathsf{BootsMap}[\mathsf{m}+1] \in \mathbb{Z}_Q$.
  Set $y' \leftarrow y - N$.
  If $y = N$,
    set $\mathbf{p}_1[1] \leftarrow -\mathsf{BootsMap}[\mathsf{m}+1]$.
  If $y \in [N+1, 2 \cdot N - 1]$,
    set $\mathbf{p}_1[N-y'+1] \leftarrow \mathsf{BootsMap}[\mathsf{m}+1] \in \mathbb{Z}_Q$.
Set $\mathsf{rotP}_0 = \sum_{i=1}^{N} \mathbf{p}_0[i] \cdot X^{i-1}$.
Set $\mathsf{rotP}_1 = \sum_{i=1}^{N} \mathbf{p}_1[i] \cdot X^{i-1}$.
Return $(\mathsf{rotP}_0, \mathsf{rotP}_1)$.

**Figure 3:** Algorithm to set up the rotation polynomial based on a look up table

**Theorem 2.** *Let $f : \mathbb{Z}_t \mapsto \mathbb{Z}_Q$ be a function with a lookup table* $\mathsf{BootsMap}$ *such that for all $x \in \mathbb{Z}_t$, we have that* $\mathsf{BootsMap}[x+1] = f(x)$. *Let* $(\mathsf{rotP}_0, \mathsf{rotP}_1) \leftarrow \mathsf{SetupPolynomial}(\mathsf{BootsMap}, N)$ *for $N \in \mathbb{N}$. Let $y = \lfloor \frac{2 \cdot N}{t} \rceil \cdot \mathsf{m} + e$, for $\mathsf{m} \in \mathbb{Z}_t$ and $e \in [-\lceil \frac{N}{t} \rceil, \lceil \frac{N}{t} \rceil]$. Then*

- $\mathsf{Coefs}(\mathsf{rotP}_0 \cdot X^y)[1] = f(\mathsf{m})$ *for $y \in [0, N-1]$, and*
- $\mathsf{Coefs}(\mathsf{rotP}_1 \cdot X^y)[1] = f(\mathsf{m})$ *for $y \in [N, 2 \cdot N - 1]$.*

## 3.3 The Functional Bootstrapping Algorithm

Our bootstrapping algorithm is depicted in Figure 4. There are three phases. First, we compute an extended ciphertext whose plaintext indicates whether the given ciphertexts phase is in $[0, N)$ or in $[N, 2 \cdot N)$. Based on this ciphertext, we build the accumulator. The final step is to blind rotate the rotation polynomial and switch the ciphertext from RLWE to LWE. Remember that the vector $\mathbf{u} \in \mathbb{Z}^u$ is such that for all $y \in \mathcal{S}$ with $\mathcal{S} \subseteq \mathbb{Z}_{2 \cdot N}$ there exists $\mathbf{x} \in \mathbb{B}^u$ such that $y = \sum_{i=1}^{u} \mathbf{x}[i] \cdot \mathbf{u}[i] \mod 2 \cdot N$. Furthermore, we denote as $\mathsf{L}_{\mathsf{RGSW}}$, $\mathsf{L}_{\mathsf{ksK}}$ and $\mathsf{L}_{\mathsf{pK}}$ the decomposition basis for RGSW samples, and two different key switch keys. Below we state the correctness theorem and devise the proof to Appendix A.

**Theorem 3** (Correctness of the Bootstrapping)**.** *Let $\mathbf{s} \in \mathbb{Z}_q^n$ for some $q, n \in \mathbb{N}$, $\mathfrak{s} \in \mathcal{R}_{N,q}$, and $\mathbf{s}_F \leftarrow$* $\mathsf{KeyExt}(\mathfrak{s})$*. Furthermore, let us define the following.*

- $\mathsf{brK} \leftarrow \mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$.

$\mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}_0, \mathsf{rotP}_1, \mathsf{L}_{\mathsf{boot}}, \mathsf{pK}, \mathsf{ksK})$

**Input:** A bootstrapping key $\mathsf{brK} = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{brK}}}(\mathfrak{s}, .)^{n\times u}$,vector $\mathbf{u} \in \mathbb{Z}^u$,
LWE sample $\mathsf{ct} = \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,q}(\mathbf{s}, .)(\mathsf{m}) = [b, \mathbf{a}^\top]^\top \in \mathbb{Z}_q^{(n+1)\times 1}$,polynomials $\mathsf{rotP}_0, \mathsf{rotP}_1, \in \mathcal{R}_{N,Q}$,
decomposition basis $\mathsf{L}_{\mathsf{boot}}$,
LWE **to** RLWE key switching key $\mathsf{pK}$,
and a LWE **to** LWE key switching key $\mathsf{ksK}$.
**Output:** A ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}},n,q}(\mathbf{s}, \Delta_{q,t'} \cdot f(\mathsf{m}))$ **for** a function $f$.

Set $\mathsf{sgnP} = 1 - \sum_{i=2}^{N} X^{i-1} \in \mathcal{R}_{N,Q}$.
Set $\mathsf{ct}_N \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q, 2\cdot N) = [b_N, \mathbf{a}_N^\top]^\top$.
Compute $\mathsf{sgnP}_b \leftarrow \mathsf{sgnP} \cdot X^{b_N} \in \mathcal{R}_{N,Q}$.
Let $\ell_{\mathsf{boot}} = \lceil \log_{\mathsf{L}_{\mathsf{boot}}} Q \rceil$.
For $i = 1$ **to** $\ell_{\mathsf{boot}}$ do,
  Set $\mathsf{acc}_i \leftarrow [\mathsf{L}_{\mathsf{boot}}^{i-1} \cdot \mathsf{sgnP}_b \cdot (\frac{\Delta_{Q,t}}{2}), 0]^\top$.
  Run $\mathsf{acc}_{\mathsf{BR},i} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_i, \mathsf{ct}_N, \mathbf{u})$.
  Run $\mathsf{acc}_{\mathbf{c},i} \leftarrow \mathsf{SampleExt}(\mathsf{acc}_{\mathsf{BR},i}, 1) + [\mathsf{L}_{\mathsf{boot}}^{i-1} \cdot \frac{\Delta_{Q,t}}{2}, \mathbf{0}]$.
Set $\mathsf{rotP}'_0 \leftarrow \mathsf{rotP}_0 \cdot X^{b_N} \in \mathcal{R}_{N,Q}$.
Set $\mathsf{rotP}'_1 \leftarrow \mathsf{rotP}_1 \cdot X^{b_N} \in \mathcal{R}_{N,Q}$.
Run $\mathsf{acc}_F \leftarrow \mathsf{BuildAcc}([\mathsf{acc}_{\mathbf{c},i}]_{i=1}^{\ell_{\mathsf{boot}}}, \mathsf{rotP}'_0, \mathsf{rotP}'_1, \mathsf{pK})$.
Run $\mathsf{acc}_{\mathsf{BR},F} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_F, \mathsf{ct}_N, \mathbf{u})$.
Run $\mathbf{c}_Q \leftarrow \mathsf{SampleExt}(\mathsf{acc}_{\mathsf{BR},F}, 1)$.
Run $\mathbf{c}_{Q,\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}_Q, \mathsf{ksK})$.
Return $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{Q,\mathsf{ksK}}, Q, q)$.

**Figure 4:** Full domain bootstrapping algorithm.

- $\mathsf{pK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{pK}}, N, 1, N, Q, \mathbf{s}_F, \mathfrak{s}, \mathsf{L}_{\mathsf{pK}})$.
- $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, N, n, 1, Q, \mathbf{s}_F, \mathbf{s}, \mathsf{L}_{\mathsf{ksK}})$.
- $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m})$.
- $\mathsf{ct}_N \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q, 2\cdot N)$, *and let us denote* $\mathsf{ct}_N \in \mathsf{LWE}_{\mathcal{B}_N,n,2\cdot N}(\mathbf{s}, \Delta_{2\cdot N,t} \cdot \mathsf{m})$.

*Finally, let* $\tilde{\mathsf{m}} \leftarrow \mathsf{Phase}(\mathsf{ct}_N)$, $\lfloor \tilde{\mathsf{m}} \rceil_t^{2\cdot N} = \mathsf{m}$ *and* $\mathsf{rotP}_0, \mathsf{rotP}_1 \in \mathcal{R}_{N,Q}$ *be such that*

- *if* $\tilde{\mathsf{m}} \in [0, N)$, *then* $\mathsf{Coefs}(X^{\tilde{\mathsf{m}}} \cdot \mathsf{rotP}_0)[1] = \Delta_{Q,t'} \cdot f(\mathsf{m})$, *and*
- *if* $\tilde{\mathsf{m}} \in [N, 2\cdot N)$, *then* $\mathsf{Coefs}(X^{\tilde{\mathsf{m}}} \cdot \mathsf{rotP}_1)[1] = \Delta_{Q,t'} \cdot f(\mathsf{m})$, *where* $f : \mathbb{Z}_t \mapsto \mathbb{Z}_{t'}$.

*If* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}_0, \mathsf{rotP}_1, \mathsf{L}_{\mathsf{boot}}, \mathsf{pK}, \mathsf{ksK})$ *and given that* $\mathcal{B}_N \leq \frac{N}{t}$, *then* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}},n,q}(\mathbf{s}, \Delta_{q,t'} \cdot f(\mathsf{m}))$, *where*

$$\mathcal{B}_N \leq \Big(\frac{2\cdot N}{q}\Big)^2 \cdot \mathcal{B}_{\mathsf{ct}} + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty,$$

*for* $N \neq 2\cdot q$, *and* $\mathcal{B}_N = \mathcal{B}_{\mathsf{ct}}$ *otherwise, and* $\mathcal{B}_{\mathsf{ct}} \leq \max(\mathcal{B}_{\mathsf{fresh}}, \mathcal{B}_{\mathsf{out}})$. *We use the bound* $\mathcal{B}_{\mathsf{fresh}}$ *if* $\mathsf{ct}$ *is a freshly encrypted ciphertext. Finally the bound* $\mathcal{B}_{\mathsf{out}}$ *of the bootstrapped* *ciphertext* $\mathsf{ct}_{\mathsf{out}}$ *that is as follows*

$$\mathcal{B}_{\mathsf{out}} \leq \frac{q^2}{Q^2} \cdot (\mathcal{B}_F + \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}) + \mathsf{Ham}(\mathbf{s}_F) \cdot ||\mathsf{Var}(\mathbf{s}_F)||_\infty,$$

*for* $Q \neq q$*, and* $\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_F + \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}$ *for* $Q = q$*, where*

- $\mathcal{B}_F \leq N \cdot \ell_{\mathsf{boot}} \cdot \mathsf{L}^2_{\mathsf{boot}} \cdot \big(\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_P\big)$
- $\mathcal{B}_{\mathsf{BR}} \leq 2 \cdot n \cdot u \cdot N \cdot \ell_{\mathsf{RGSW}} \cdot \mathsf{L}^2_{\mathsf{RGSW}} \cdot \mathcal{B}_{\mathsf{brK}},$
- $\mathcal{B}_{KS} \leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}^2_{\mathsf{ksK}} \cdot \mathcal{B}_{\mathsf{ksK}},$ *and*
- $\mathcal{B}_P \leq N \cdot \ell_{\mathsf{pK}} \cdot \mathsf{L}^2_{\mathsf{pK}} \cdot \mathcal{B}_{\mathsf{pK}}.$

## 3.4 Application of Functional Bootstrapping

Below we discuss a few simple tricks on how to use our bootstrapping algorithm efficiently. We stress that the bootstrapping can compute any function over $\mathbb{Z}_t$ for an arbitrary modulus $t \in \mathbb{N}$. Furthermore, we can keep computing very efficient affine functions on bootstrapped Regev ciphertexts before the next bootstrapping operation.

**Computing in $\mathbb{Z}_t$.** Note that with the full domain functional bootstrapping algorithm we can correctly compute affine functions without the need to bootstrap immediately. Specifically, we use the linear homomorphism of LWE samples (see Lemma 1) to compute on the ciphertext $\mathbf{c}_Q$ at Step 13 of $\mathsf{Bootstrap}$ at Figure 4. In Section 4 we evaluate correctness for this particular setting as in this case computing the linear homomorphism will not increase the error that stems from the key switching key.

To compute the product $x \cdot y$, we compute $\big(\frac{(x+y)}{2}\big)^2 - \big(\frac{(x-y)}{2}\big)^2$. Therefore evaluating the product requires only two functional bootstrapping invocations, we compute the square, and the addition/subtraction induces only a small error when done outside the bootstrapping procedure. Similarly, we can compute $\max(x, y)$ by $\max(x - y, 0) + y$, which costs only one functional bootstrapping.

To summarize, since we can compute all functions over $\mathbb{Z}_t$ with a single bootstrapping operation and efficiently compute affine functions on plaintexts in $\mathbb{Z}_t$, we can compute arithmetic circuits complemented with gates beyond only addition and multiplication. For example, we can compute the multiplicative inverse modulo $t$ needed for Gaussian elimination and solve systems of linear equations homomorphically. Note that FHE schemes like BGV/BFV [BV11, Bra12, FV12, BGV12] can only compute addition and multiplication. Hence to compute a multiplicative inverse with BGV/BFV, we need to represent the computation as a boolean circuit. With FDFB, we can compute the inverse with just one bootstrapping operation.

**Multiple Univariate Functions With the same Input.** For efficiency it is important to note, that we might amortize the cost of computing multiple functions on a encrypted ciphertext. After the first bootstrapping operation, we may reuse the $\mathsf{acc}_{\mathbf{c},i}$ values to compute the next function on the same input. This observation is particularly important for the decomposition of the encrypted integers. Namely, we can binary decompose a field element and then compose binary vector back to integers.

**Extending the Arithmetic.** Finally, we can leverage the Chinese Reminder theorem and the fact that the ring $\mathbb{Z}_t$ for $t = \prod_{i=1}^m t_i$ where the $t_i$ are pairwise co-prime is isomorphic to the product ring $\mathbb{Z}_{t_1} \times \cdots \times \mathbb{Z}_{t_m}$ as a powerful tool to extend the size of the arithmetic. This way we can handle modular arithmetic for a large composite modulus, while having cheap leveled linear operations, and one modular multiplication per factor. Note that we can compute arbitrary polynomials in $\mathbb{Z}_t$ in the CRT representation. To set the rotation polynomials for each component of the ciphertext vector we do as follows. For each number $x \in \mathbb{Z}_t$, we compute its CRT representation $x_1, \ldots, x_m$. Then we compute the polynomial

**Table 2: Parameter Sets.** The modulus $Q_1 = 2^{62} - 65535$, $Q_2 = 2^{63} - 278527$, $Q_3 = 2^{48} - 163839$, and $Q_4 = 2^{32} - 139263$. For all parameters sets we set the error variance of fresh (R)LWE samples to 10.24 which corresponds to standard deviation 3.2. The key $\mathbf{s}$ is always binary; thus $u = 1$.

| Set | FDFB:80:7 | FDFB:100:7 | FDFB:80:8 | FDFB:100:8 | TFHE:100:7 | TFHE:80:2 | TFHE:100:2 |
|---|---|---|---|---|---|---|---|
| $n$ | 700 | 1100 | 700 | 1100 | 1500 | 424 | 525 |
| $N$ | $2^{12}$ | $2^{12}$ | $2^{13}$ | $2^{13}$ | $2^{12}$ | $2^{10}$ | $2^{10}$ |
| $q$ | $2^{13}$ | $2^{13}$ | $2^{14}$ | $2^{14}$ | $2^{13}$ | $2^{11}$ | $2^{11}$ |
| $Q$ | $Q_2$ | $Q_2$ | $Q_2$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_4$ |
| $\mathsf{L}_{\mathsf{boot}}$ | $2^{11}$ | $2^{11}$ | $2^{8}$ | $2^{8}$ | N/A | N/A | N/A |
| $\mathsf{L}_{\mathsf{RGSW}}$ | $2^{9}$ | $2^{9}$ | $2^{9}$ | $2^{9}$ | $2^{16}$ | $2^{11}$ | $2^{11}$ |
| $\mathsf{L}_{\mathsf{ksK}}$ | $2^{4}$ | 2 | $2^{4}$ | 2 | 2 | $2^{4}$ | $2^{4}$ |
| $\mathsf{L}_{\mathsf{pK}}$ | $2^{13}$ | $2^{13}$ | $2^{13}$ | $2^{13}$ | N/A | N/A | N/A |
| stddev | $2^{39}, 3.2$ | $2^{41}, 3.2$ | $2^{39}, 3.2$ | $2^{41}, 3.2$ | $2^{18}, 3.2$ | $2^{16}, 3.2$ | $2^{16}, 3.2$ |
| $\mathbf{s}$ | $[0,1], 64$ | $[0,1], 64$ | $[0,1], 64$ | $[0,1], 64$ | $[0,1], 64$ | $[0,1], -$ | $[0,1], -$ |
| $\mathfrak{s}$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ | $\mathbb{Z}_Q^N, -$ |

$y = P(x)$ and $y$'s CRT representation $[y_i]_{i=1}^m$. We set the rotation polynomials for the $i$th bootstrapping algorithm to map $x_i$ to $y_i$, as described in Section 3.2. Furthermore, additions and scalar multiplications can still be performed without bootstrapping.

### 3.4.1 Computing Negacyclic Functions.

To compute negacyclic functions, we use the TFHE bootstrapping algorithm given by Definition 13. Note that FDFB contains all the necessary public key material to use the standard TFHE bootstrapping algorithm, which is faster and sufficient for certain applications. Therefore we may view FDFB as an extension of TFHE rather than a substitute. While the algorithm is from [CGGI16], we show by Theorem 1 that, assuming the plaintext is in $[0, t/2)$, the outcome of the bootstrapping is determined by the message and the rotation polynomial. In contrast to full domain bootstrapping, in this method, we must be careful when performing leveled operations. In particular, when the message exceeds the interval $[0, t/2)$, then the outcome of the bootstrapping may be incorrect.

## 4 Parameters, Performance, and Tests

In the next paragraph, we describe our parameter sets and discuss the strategy for how we chose the parameters. Furthermore, we give the security and correctness estimates for all proposed parameter sets. Then, we discuss the details of our implementation and give the results of our microbenchmarks. Finally, we describe our example applications and give the results of the performance tests for those applications.

**Parameter Sets.** Table 2 shows our different parameter sets, Table 4 gives correctness for each parameter set and the size of the message space, Table 5 contains an overview of the ciphertext size, and time to run a single bootstrapping operation, and Table 3 gives the security estimations that we obtained when running the LWE-estimators [APS15, CSY21].

We denote the parameter sets FDFB:$s$:$t$ and TFHE:$s$:$t$ where $s$ is the security level, and $t$ is the number of bits of the plaintext space for which the probability of having an incorrect outcome is very low. For example, FDFB:100:7 offers 100 bits of security and is targeted at 7-bit plaintexts. We stress that the targeted plaintext space does not exclude larger plaintext moduli. In particular, according to Table 4, all parameter sets can handle larger plaintexts, but the probability that the least significant bit of the input message changes obviously grows with the plaintext space size, shifting the scheme from FHE towards homomorphic encryption for approximate arithmetic.

First, we note that if the ratio $Q/q$ is high, then the error after modulus switching from $Q$ to $q$ is dominated by the Hamming weight of the LWE secret key $\mathbf{s}$. To minimize the error, we choose $q = 2 \cdot N$ to avoid another modulus switching. To be able to compute

**Table 3: Security estimations for the parameter sets.** This table contains the the security estimates obtained by running the LWE estimator [APS15] for RLWE and LWE samples. The numbers represent base two logarithms of estimated required ring operations to lunch a sucessfull attack.

| Set | RLWE | | | LWE | | | |
|---|---|---|---|---|---|---|---|
| | uSVP | dec | dual | uSVP | dec | dual | Hyb. Primal |
| FDFB:80:7 | 282.4 | 306.1 | 294.7 | 87.8 | 309.5 | 82.7 | 81.6 |
| FDFB:100:7 | 282.4 | 306.1 | 294.7 | 138.2 | 644.3 | 129.0 | 100.0 |
| FDFB:80:8 | 548.5 | 596.2 | 570.1 | 87.8 | 309.5 | 82.7 | 81.6 |
| FDFB:100:8 | 548.5 | 596.2 | 570.1 | 138.2 | 644.3 | 129.0 | 100.0 |
| TFHE:100:7 | 236.9 | 255.5 | 247.4 | 139.6 | 322.7 | 128.9 | 100.8 |
| TFHE:80:2 | 116.9 | 128.3 | 125.3 | 81.3 | 207.3 | 86.2 | 88.9 |
| TFHE:100:2 | 116.9 | 128.3 | 125.3 | 100.0 | 264.3 | 107.1 | 111.0 |

**Table 4: Correctness.** The table contains upper bounds on the probability to have an error in the outcome of the bootstrapping algorithms. For our full domain bootstrapping, we give two values. The first is the upper bound on the error probability for ciphertexts after bootstrapping. The second value is the upper bound for a ciphertext after bootstrapping and computing an affine function of size 784. That is, a function of the form $w_0 + \sum_{i=1}^{784} x_i \cdot w_i \in \mathbb{Z}_t$ where $x_i$ are the input plaintexts, and the $w_i$'s are the weights. We emphasize that the probabilities are upper-bounds, and the empirical correctness errors are much lower.

| Set / $\log_2(t)$ | 6 | 7 | 8 | 9 | 10 | 11 | Key |
|---|---|---|---|---|---|---|---|
| FDFB:80:7 | 0.0, 0.0 | $2^{-31}$, $2^{-21}$ | $2^{-9}$, 0.07 | 0.12, 0.61 | 0.44, 0.89 | 0.69, 0.97 | 95% |
| FDFB:100:7 | 0.0, 0.0 | $2^{-31}$, $2^{-18}$ | $2^{-9}$, 0.13 | 0.12, 0.67 | 0.44, 0.91 | 0.69, 0.97 | 95% |
| FDFB:80:8 | 0.0, 0.0 | 0.0, 0.0 | $2^{-26}$, $2^{-17}$ | $2^{-8}$, 0.12 | 0.16, 0.66 | 0.48, 0.90 | 67% |
| FDFB:100:8 | 0.0, 0.0 | 0.0, 0.0 | $2^{-26}$, $2^{-13}$ | $2^{-8}$, 0.23 | 0.16, 0.75 | 0.48, 0.93 | 67% |
| TFHE:100:7 | 0.0 | $2^{-24}$ | $2^{-7}$ | 0.18 | 0.50 | 0.73 | 99% |
| TFHE:80:2 | 0.57 | 0.77 | 0.88 | 0.94 | 0.97 | 0.98 | 14% |
| TFHE:100:2 | 0.58 | 0.78 | 0.89 | 0.94 | 0.97 | 0.98 | 17% |

number theoretic transforms we choose $Q$ prime and $Q = 1 \mod 2 \cdot N$. An important observation is that the error does not depend on the RLWE secret key $\mathfrak{s}$. Hence, we choose $\mathfrak{s}$ from the uniform distribution over $\mathcal{R}_{N,Q}$. For all RLWE samples, we set the standard deviation of the error as 3.2. We choose the LWE secret key $\mathbf{s}$ from $\{0, 1\}^n$ such that the Hamming weight of the vector is 64. Recall that LWE with binary keys is asymptotically as secure as the standard LWE [Mic18], and is also used in many implementations of TFHE. The Hamming weight is the same as used in the HElib library [HS14, HS15, Alb17]. To compensate for the loss of security due to a small Hamming weight, we choose the standard deviation of LWE samples error much higher that the in case of RLWE. For binary components of $\mathbf{s}$ we have $||\mathbf{u}|| = 1$ which is as in TFHE [CGGI16, CGGI20]. We choose the decomposition bases $\mathsf{L}_{\mathsf{boot}}$, $\mathsf{L}_{\mathsf{RGSW}}$, and $\mathsf{L}_{\mathsf{pK}}$ to minimize the amount of computation. Due to the higher noise variance of the LWE samples in the key switching key, we choose the decomposition basis $\mathsf{L}_{\mathsf{ksK}}$ relatively low, but we note that in terms of efficiency, key switching does not contribute much.

To evaluate security, we use the LWE estimator [APS15] commit `fb7deba` and the Sparse LWE estimator [CSY21] commit `6ab7a6e`. The LWE Estimator estimates the security level of a given parameter set by calculating the complexity of the meet in the middle exhaustive search algorithm, Coded-BKW [GJS15], dual-lattice attack and small/sparse secret variant [Alb17], lattice-reduction + enumeration [LP11], primal attack via uSVP [AFG14, BG14] and Arora-Ge algorithm [AG11] using Gröbner bases [ACFP14]. The sparse LWE estimator is build on top of [APS15], but additionally considers a variant [SC19] of Howgrave-Graham's hybrid attack [How07].

We calculate the correctness by taking the standard deviation of the bootstrapped sample. In particular, given that $\mathcal{B}$ is the bound on the variance of the error we set $\mathsf{stddev}_{\mathcal{B}} =$

**Table 5: Complexity for the parameter sets.** We computed the size of the keys and ciphertexts by taking the byte size of every field element and counting the number of field elements.

| Set | Time [ms] | PK [MB] | CT [KB] |
|---|---|---|---|
| FDFB:80:7 | 9278 | 2351.95 | 1.40 |
| FDFB:100:7 | 13781 | 4624.31 | 2.20 |
| FDFB:80:8 | 33117 | 7388 | 1.40 |
| FDFB:100:8 | 45266 | 14318 | 3.00 |
| TFHE:100:7 | 7757 | 921.91 | 2.43 |
| TFHE:80:2 | 306 | 31.72 | 0.72 |
| TFHE:100:2 | 321 | 25.62 | 0.58 |

**Table 6: Basic Performance Tests.** The table contains timings for modular Addition (Add), Scalar Multiplication (S. Mul) and Multiplication (Mul). All timings in seconds. We consider the timings for operations from $\mathbb{Z}_{2^k} \times \mathbb{Z}_{2^k}$ to $\mathbb{Z}_{2^k}$ for $k \in \{7, 8\}$.

| Param. | Add | S.Mul | Mul |
|---|---|---|---|
| FDFB:80:7 | $1 \cdot 10^{-6}$ | $4 \cdot 10^{-6}$ | 18 |
| FDFB:100:7 | $1 \cdot 10^{-6}$ | $7 \cdot 10^{-6}$ | 26 |
| TFHE:80:2 | 15.6 | 15.6 | 15.6 |
| TFHE:100:2 | 19.2 | 19.2 | 19.2 |
| FDFB:80:8 | $1 \cdot 10^{-6}$ | $3 \cdot 10^{-6}$ | 66 |
| FDFB:100:8 | $1 \cdot 10^{-6}$ | $4 \cdot 10^{-6}$ | 89 |
| TFHE:80:2 | 18 | 18 | 18 |
| TFHE:100:2 | 21 | 21 | 21 |

$\sqrt{\mathcal{B}}$, and compute the probability of a correct bootstrapping as $\mathsf{erf}\left(\frac{q}{2 \cdot t \cdot \mathsf{stddev}_{\mathcal{B}} \cdot \sqrt{2}}\right)$. In Table 4 for the sake of readability, we approximate the probability of getting an incorrect output as a power of 2. That is we find the smallest $i \in \mathbb{N}$ such that $2^{-i} \leq 1 - \mathsf{erf}\left(\frac{q}{2 \cdot t \cdot \mathsf{stddev}_{\mathcal{B}} \cdot \sqrt{2}}\right)$. For the FDFB sets we show correctness calculated from the bound $\mathcal{B}_N$ as given in Theorem 3. Furthermore, we show correctness when computing affine functions after bootstrapping. For TFHE we take the error bound $\mathcal{B}_N$ as given by Theorem 1. In both cases, according to our parameter selection, we have that $\mathcal{B}_N$ is computed from $\mathcal{B}_{\mathsf{ct}}$ since it is higher than $\mathcal{B}_{\mathsf{fresh}}$. Note that $\mathcal{B}_{\mathsf{ct}}$ is a ciphertext output by a previous bootstrapping.

**Implementation.** We ran our experiments on a virtual machine with 60 cores, featuring 240GB of RAM and using Intel Xeon Cascade lake processors, and the measurements are averaged over five runs. We implemented our bootstrapping technique by modifying the latest version of the PALISADE [PAL21] library, , using the Intel HEXL [BKS+21] library, as it is already featured in the implementation of [DM14] and [CGGI16]. Note that the timings in PALISADE for TFHE are different than reported in [CGGI16]. This is mostly due to an optimization in TFHE that chooses the modulus $Q$ as $Q = 2^{32}$ or $Q = 2^{64}$ and uses the natural modulo reduction of unsigned int and long C++ types. Then instead of computing the number theoretic transform, TFHE computes the fast Fourier transform. We note that the same optimization is possible for our algorithms.

**Evaluation.** Table 6 shows the efficiency of modular addition, scalar multiplication and multiplication targeting different plaintext spaces, respectively, for our parameter sets. We compare our approach to operations on binary ciphertexts. To this end, we use the vertical look-up tables from [CGGI17], as they turned out to be faster than a circuit-based implementation. Note that, while vertical LUTs are significantly faster than the horizontal tables, which were also introduced in [CGGI17], they can compute at most one function at the same time. Therefore multiple tables need to be maintained for each output bit, requiring additional memory. We can observe that for modular multiplication, our approach is less efficient than the LUT based one. However, since we can use the linear homomorphism of LWE samples, our techniques are especially well suited to addition and scalar multiplication.

To evaluate a neural network on encrypted inputs, we discretized two neural networks that recognize handwritten digits from the MNIST dataset. Note that the discretization of a floating-point neural network to integer models with modular arithmetic is a delicate process. Many operations such as normalization cannot be applied in the same manner, and therefore the discretization process often results in a drop of accuracy. Do discretize our neural networks we use a parameter $\delta$ called the discretization factor. The smaller the discretization factor, the smaller the accuracy. Increasing the discretization factor requires handling larger plaintexts. Hence we have a tradeoff between the accuracy of the discretized NN and the plaintext space of the FHE scheme. The configurations and accuracies are given at Table 9. Note that for our experiments, the peak is around an eight and 9-bit modulus and falls for smaller or larger modulus. For a smaller modulus, it is quite obvious that the precision is simply too small. We can explain the decrease in accuracy for larger plaintext moduli by the error from homomorphic evaluation. In particular, according to Table 4, bootstrapping, for example, an 11-bit plaintext, will be incorrect with a rather high probability. Hence the drop in accuracy.

In what follows, we give the notation used to specify the neural networks. We always denote the weights with $w$'s and biases with $b$'s. By $\mathsf{Dense}{:}d{:}f$ we denote a dense layer that on input a vector $[x_i]_{i=1}^m$ outputs a vector $\left[f(b_j + \sum_{i=1}^m x_i \cdot w_{j,i})\right]_{j=1}^d$.

Convolution layers are denoted as $\mathsf{Conv2D}{:}h{:}(d,d){:}f$, where $h$ is the number of filters, $(d,d)$ specify the shape of the filters and $f$ is an activation function. A $\mathsf{Conv2D}$ layer takes as input a tensor $[x_{g,i,j}]_{g=1,i=1,j=1}^{c,m,l}$ and outputs

$$\left[\left[f\big(b_{i',j'}^{(k)} + \sum_{g=1,i=1,j=1}^{c,d,d} x_{g,i+i',j+j'} \cdot w_{g,i,j}^{(k)}\big)\right]_{i'=0,j'=0}^{m-d,l-d}\right]_{k=1}^{h}.$$

Average pooling $\mathsf{AvgPool2D}{:}(d,d)$ on input a tensor $[x_{g,i,j}]_{g=1,i=1,j=1}^{c,m,l}$ outputs

$$\left[\left[\frac{\sum_{i=1,j=1}^{d,d} x_{g,i+i',j+j'}}{d^2}\right]_{i'=0,j'=0}^{m-d,l-d}\right]_{g=1}^{c}$$

Finally, $\mathsf{Softmax}{:}d$ takes as input a vector $[x_i]_{i=1}^m$, computes $\mathbf{z} = \left[b_j + \sum_{i=1}^m x_i \cdot w_{j,i}\right]_{j=1}^d$ and outputs $\left[\mathsf{Softmax}(\mathbf{z}, i) = \frac{e^{\mathbf{z}[i]}}{\sum_j^d e^{\mathbf{z}[j]}}\right]_{i=1}$.

We compared FDFB with a functionally equivalent implementation based on vertical LUTs [CGGI17]. In particular, for activation functions, we run $k$, $k$-to-1 bit LUTs for $k \in \{6,7,8,9,10,11\}$ and to compute addition and scalar multiplication for the affine function, we must use $k$, $2k$-to-1 bit LUTs. We note that we also tested the method of Carpov et al. [CIM19] to compute the activation function on set $\mathsf{TFHE}{:}100{:}7$. However, the timing to evaluate the affine function in the LUT-based method is so overwhelming that we did not notice any significant difference. Furthermore, we note that for the last layer, we chose a bigger plaintext modulus and compute the activation function in plaintext, that is we decrypt the result of the affine function before computing the softmax activation. We include the evaluation accuracies for plaintext spaces that exceed the target space. Such settings are extremely useful for computations in which a small amount of error is acceptable, such as activation functions in neural network inference. We can see that our approach significantly outperforms LUT-based techniques. The most important reason for this is that we can compute affine functions within neurons almost instantly due to the linear homomorphism of LWE samples. Binary ciphertext requires the evaluation of

**Table 7: Performance for evaluating neural networks.** The table contains the times to evaluate the MNIST-1-$f$ and MNIST-2-$f$ neural networks. The time to evaluate the neural networks is given in hours. Due to extremely high execution times, we only present estimates for TFHE, which we base on Table 6.

| | Evaluation Time MNIST-1-$f$ | | | | | |
|---|---|---|---|---|---|---|
| Param. / $\log_2(t)$ | 6 | 7 | 8 | 9 | 10 | 11 |
| FDFB:80:7 | 0.08 | | | | | |
| FDFB:100:7 | 0.35 | | | | | |
| FDFB:80:8 | 0.17 | | | | | |
| FDFB:100:8 | 0.76 | | | | | |
| TFHE:80:2 | 99.24 | 135.88 | 177.28 | 225.84 | 289.82 | 392.14 |
| TFHE:100:2 | 122.18 | 165.9 | 215.44 | 274.44 | 351.42 | 460.94 |
| | Evaluation Time MNIST-2-$f$ | | | | | |
| FDFB:80:7 | 0.37 | | | | | |
| FDFB:100:7 | 0.65 | | | | | |
| FDFB:80:8 | 1.1 | | | | | |
| FDFB:100:8 | 1.8 | | | | | |
| TFHE:80:2 | 24.56 | 33.64 | 43.88 | 55.9 | 71.74 | 97.08 |
| TFHE:100:2 | 33.64 | 41.06 | 53.3 | 67.94 | 86.98 | 114.1 |

.

**Table 8: Accuracy for homomorphic evaluation of the neural networks.** For accuracy we show values for $f = \text{ReLU}$, but stress that we can use arbitrary univariate functions without impacting execution time. We discretized the models using a discretization factor $\delta = 6$ for MNIST-1-$f$ and $\delta = 4$ for MNIST-2-$f$.

| | Accuracy MNIST-1 | | | | | |
|---|---|---|---|---|---|---|
| Param. / $\log_2(t)$ | 6 | 7 | 8 | 9 | 10 | 11 |
| FDFB:80:7 | 0.90 | 0.90 | 0.95 | 0.95 | 0.90 | 0.70 |
| FDFB:100:7 | 0.90 | 0.90 | 0.95 | 0.90 | 0.90 | 0.80 |
| FDFB:80:8 | 0.85 | 0.90 | 0.95 | 0.90 | 0.95 | 0.90 |
| FDFB:100:8 | 0.80 | 0.95 | 0.95 | 0.95 | 0.95 | 0.90 |
| TFHE:80:2 | 0.82 | 0.94 | 0.95 | 0.95 | 0.95 | 0.95 |
| TFHE:100:2 | 0.82 | 0.94 | 0.95 | 0.95 | 0.95 | 0.95 |
| | Accuracy MNIST-2 | | | | | |
| Param. / $\log_2(t)$ | 6 | 7 | 8 | 9 | 10 | 11 |
| FDFB:80:7 | 0.15 | 0.25 | 0.85 | 0.90 | 0.90 | 0.90 |
| FDFB:100:7 | 0.15 | 0.30 | 0.85 | 0.95 | 0.90 | 0.85 |
| FDFB:80:8 | 0.15 | 0.30 | 0.85 | 0.90 | 0.95 | 0.90 |
| FDFB:100:8 | 0.15 | 0.30 | 0.90 | 0.90 | 0.95 | 0.95 |
| TFHE:80:2 | 0.14 | 0.22 | 0.57 | 0.93 | 0.93 | 0.93 |
| TFHE:100:2 | 0.14 | 0.22 | 0.57 | 0.93 | 0.93 | 0.93 |

**Table 9: Neural Networks Specification.** The table contains specifications for both models that we train and evaluate and the baseline accuracy of the non-discretised model using floating point numbers. The "input" row gives the dimension of the inputs. The last row reports on the accuracy of the neural network. Other rows specify the given layer of the neural network.

| | MNIST-1-$f$ | MNIST-2-$f$ |
|---|---|---|
| Input | $[x_i]_{i=1}^{784}$ | $[x_i]_{h=1 i=1,j=1}^{1,28,28}$ |
| 1 | Dense:784:$f$ | Conv2D:4:(6,6):$f$ |
| 2 | Dense:510:$f$ | AvgPool2D:(2,2) |
| 3 | Softmax:10 | Conv2D:16:(6,6):$f$ |
| 4 | | AvgPool2D:(2,2) |
| 5 | | AvgPool2D:16:(3,3):$f$ |
| 6 | | Dense:64:$f$ |
| 7 | | Softmax:10 |
| Baseline accuracy for ReLU$(x) = \max(0, x)$ | 0.96 | 0.98 |

.

expensive LUT at each step. In the case of the LUT-based approach, we note that the computation will be exact. Therefore, the accuracy will be identical to the discretized neural network with applied modulo reduction. Table 7 shows the time required to evaluate the whole neural network. We test our network on 20 samples for each plaintext space and achieve the best speedups for sets FDFB:80:7 and FDFB:100:7 that outperform the LUT-based technique by a factor of over 1200 and 282. Even in the worst case with the MNIST-2-$f$ network, for which the affine functions are significantly smaller, we obtain speedups 44.4 and 29.6 for FDFB:80:8 against TFHE:80:2, and respectively FDFB:100:8 against TFHE:100:2 for 8-bit plaintexts. By computing the activation functions of the network over 60 cores, we manage to lower the evaluation significantly. To estimate the speedup one would gain by parallelizing the LUT-based technique, we averaged the speedup gained by the FDFB and applied it to the LUT-based timings. We did not parallelize our bootstrapping technique itself. However, we note that there is a lot of room to do so.

Finally, let us recall that [BMMP18] tests neural network inference using TFHE and achieves very good timings and accuracies to classify the MNIST dataset. They report evaluating a network with 30 neurons in 0.49 seconds and achieving 93.71% accuracy, and with 100 neurons in 1.65 seconds and achieving 96.35% accuracy. For classification, they trained two neural networks, one with 30 neurons and one with 100, where the activation function was implemented by the sign function. Both networks had one hidden layer. Importantly note that the sign function is already negacyclic, which allows us to use the TFHE bootstrapping to compute the activation functions. Note that, especially the reported accuracies are very impressive since, as observed in [BMMP18], the chosen parameters had quite low correctness, and the authors observed many incorrect bootstrapping outcomes, yet the network is robust enough to achieve accuracy at a level that our (larger) network achieves in plaintext. In [CJP21] the authors claim to evaluate neural networks with 20, 50, and 100 layers that classify MNIST in time around one minute, depending on the network or parameter set used. In each case, the authors report accuracies that are usually over 90%. In this case, the neural networks had ReLU activation functions. We refer to [CJP21] for the reported timings and accuracies. While we did not replicate the result or establish all the relevant parameters of their TFHE instantiation, we note that FDFB, in principle, has the capability to achieve similar results. As described in Section 3.4, FDFB has all the required key materials necessary to run the TFHE bootstrapping algorithm used in both works [BMMP18, CJP21].

# 5 Conclusion

We believe that our evaluations showed that it is practically feasible to evaluate arithmetic circuits over encrypted data using fully homomorphic encryption. We showed that parallelization might play a pivotal role in further reducing the timing. Therefore, an open question is how much time can be improved when exploiting graphics processing units or larger clusters. Finally, we believe that our method may either be a complementary algorithm to CKKS methods or a competitive alternative that gives the evaluation's exact results instead of approximate.

**Acknowledgements.** This work has been partially funded/supported by the German Ministry for Education and Research through funding for the project CISPA-Stanford Center for Cybersecurity (Funding number: 16KIS0927).

# References

[ABSdV19] Mark Abspoel, Niek J. Bouman, Berry Schoenmakers, and Niels de Vreede. Fast secure comparison for medium-sized integers and its application in binarized neural networks. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 453–472. Springer, Heidelberg, March 2019.

[ACFP14] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. Algebraic algorithms for LWE. Cryptology ePrint Archive, Report 2014/1018, 2014. https://eprint.iacr.org/2014/1018.

[AFG14] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13*, volume 8565 of *LNCS*, pages 293–310. Springer, Heidelberg, November 2014.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

[Alb17] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

[AP13] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2013.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[Bar89] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150 – 164, 1989.

[BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 217–251. Springer, Heidelberg, May 2018.

[BG14] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 322–337. Springer, Heidelberg, July 2014.

[BGGJ18] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758, 2018. https://eprint.iacr.org/2018/758.

[BGPG20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[BKS+21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. Intel HEXL (release 1.2). https://github.com/intel/hexl, 2021.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Heidelberg, August 2018.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014*, pages 1–12. ACM, January 2014.

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 395–412. ACM Press, November 2019.

[CdWM+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035, 2017. https://eprint.iacr.org/2017/035.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Heidelberg, December 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

[CH18] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Heidelberg, April / May 2018.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 106–126. Springer, Heidelberg, March 2019.

[CJL+20] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, volume 15, 2020.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Heidelberg, December 2017.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. Cryptology ePrint Archive, Report 2021/729, 2021. https://eprint.iacr.org/2021/729.

[CSY21] Jung Hee Cheon, Yongha Son, and Donggeon Yhee. Practical FHE parameters against lattice attacks. Cryptology ePrint Archive, Report 2021/039, 2021. https://eprint.iacr.org/2021/039.

[DGBL+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. ICML'16, page 201–210. JMLR.org, 2016.

[DM14] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. https://eprint.iacr.org/2014/816.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, April 2015.

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

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, Feb. 2021.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH+19] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, and Daniele Micciancio. Homomorphic encryption for finite automata. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 473–502. Springer, Heidelberg, December 2019.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 23–42. Springer, Heidelberg, August 2015.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing messages and optimizing bootstrapping in GSW-FHE. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 699–715. Springer, Heidelberg, March / April 2015.

[HEl] HElib. https://github.com/homenc/HElib.

[How07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, Heidelberg, August 2007.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2014.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 641–670. Springer, Heidelberg, April 2015.

[HS21] Shai Halevi and Victor Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, January 2021.

[ISZ19] Malika Izabachène, Renaud Sirdey, and Martin Zuber. Practical fully homomorphic encryption for fully masked neural networks. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *CANS 19*, volume 11829 of *LNCS*, pages 24–36. Springer, Heidelberg, October 2019.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1209–1222. ACM Press, October 2018.

[jLHH+21] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 2021.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1651–1669. USENIX Association, August 2018.

[Klu22] Kamil Kluczniak. NTRU-$\nu$-um: Secure fully homomorphic encryption from NTRU with small modulus. Cryptology ePrint Archive, Report 2022/089, 2022. https://eprint.iacr.org/2022/089.

[KSK+18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):23–31, 2018.

[KSK+20] Duhyeong Kim, Yongha Son, Dongwoo Kim, Andrey Kim, Seungwan Hong, and Jung Hee Cheon. Privacy-preserving approximate gwas computation based on homomorphic encryption. *BMC Medical Genomics*, 13(7):1–12, 2020.

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 619–631. ACM Press, October / November 2017.

[LM20] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. Cryptology ePrint Archive, Report 2020/1533, 2020. https://eprint.iacr.org/2020/1533.

[LMP21] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. Cryptology ePrint Archive, Report 2021/1337, 2021. https://eprint.iacr.org/2021/1337.

[LMSS22] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. Cryptology ePrint Archive, Report 2022/816, 2022. https://eprint.iacr.org/2022/816.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011.

[LW20] Changmin Lee and Alexandre Wallet. Lattice analysis on MiNTRU problem. Cryptology ePrint Archive, Report 2020/230, 2020. https://eprint.iacr.org/2020/230.

[Mic18] Daniele Micciancio. On the hardness of learning with errors with binary secrets. *Theory of Computing*, 14(13):1–17, 2018.

[MS18] Daniele Micciancio and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl, July 2018.

[PAL21] PALISADE Lattice Cryptography Library (release 1.11.5). https://palisade-crypto.org/, 2021.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[RSC+19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1501–1518. USENIX Association, August 2019.

[SC19] Yongha Son and Jung Hee Cheon. Revisiting the hybrid attack on sparse and ternary secret LWE. Cryptology ePrint Archive, Report 2019/1019, 2019. https://eprint.iacr.org/2019/1019.

[Sch] Leonard Schild. FDFB Implementation. https://github.com/cispa/full-domain-functional-bootstrap.

[SEA22] Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL, March 2022. Microsoft Research, Redmond, WA.

[YXS+21] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347, 2021. https://eprint.iacr.org/2021/1347.

# A Correctness of the Bootstrapping

We give the proofs of the lemmas and theorem from our paper. For readability sake, we recall the lemmas and theorems.

**Lemma 9** (Correctness of Public Mux)**.** *Let* $[\mathbf{c}_i]_{i=1}^{\ell}$ *be a vector where* $\mathbf{c}_i \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{c}},n,N,q}(\mathbf{s}, \mathsf{m} \cdot \Delta_{q,t} \cdot \mathsf{L}^{i-1})$ *where* $n, N, q \in \mathbb{N}$, $\mathsf{m} \in \{0, 1\}$, $\ell = \lceil \log_{\mathsf{L}} q \rceil$. *Let* $\mathfrak{p}_0$, $\mathfrak{p}_1 \in \mathcal{R}_{N,q}$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{PubMux}([\mathbf{c}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1)$, *then* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{GLWE}_{\mathcal{B}_{\mathsf{out}},n,N,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$, *where* $\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{c}}$. *If* $\mathfrak{p}_0$, $\mathfrak{p}_1$ *are monomials then* $N$ *disappears from the above inequality.*

*Proof.* From the correctness of linear homomorphism of GLWE samples, we have

$$\begin{aligned}\mathbf{c}_{\mathsf{out}} &= \begin{bmatrix}\Delta_{q,t}\mathfrak{p}_0 \\ \mathbf{0}\end{bmatrix} + \sum_{i=1}^{\ell} \mathbf{p}[i] \cdot \mathsf{GLWE}_{\mathcal{B}_{\mathbf{c}},n,N,q}(\mathbf{s}, \mathsf{m}\Delta_{q,t}\mathsf{L}^{i-1}) \\ &= \begin{bmatrix}\Delta_{q,t}\mathfrak{p}_0 \\ \mathbf{0}\end{bmatrix} + \mathsf{GLWE}_{\mathcal{B}_{\mathsf{out}},n,N,q}(\mathbf{s}, \Delta_{q,t}\mathsf{m}(\mathfrak{p}_1 - \mathfrak{p}_0)) \\ &= \mathsf{GLWE}_{\mathcal{B}_{\mathsf{out}},n,N,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}}),\end{aligned}$$

where $\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{c}}$. If $\mathfrak{p}_0$, $\mathfrak{p}_1$ are monomials then $N$ disappears from the above inequality. □

**Lemma 10** (Correctness of the Accumulator Builder)**.** *Let* $[\mathsf{acc}_i]_{i=1}^{\ell}$ *be a vector of LWE samples* $\mathsf{acc}_i \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{acc}},n,q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m} \cdot \mathsf{L}^{i-1})$, *where* $\mathsf{m} \in \{0, 1\}$, $n, q \in \mathbb{N}$, $\mathcal{B}_{\mathsf{acc}} \leq q$, $\mathbf{s} \in \mathbb{Z}_q^{n \times 1}$, *and* $\ell = \lceil \log_{\mathsf{L}} q \rceil$. *Let* $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, n, 1, N, q, \mathbf{s}, \mathfrak{s}, \mathsf{L}_{\mathsf{ksK}})$, *where* $N \in \mathbb{N}$, $\mathfrak{s} \in \mathcal{R}_{N,q}$ *and* $\mathsf{L}_{\mathsf{ksK}} \leq q$. *Finally let* $\mathfrak{p}_0, \mathfrak{p}_1 \in \mathcal{R}_{N,q}$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{BuildAcc}([\mathsf{acc}_i]_{i=1}^{\ell}, \mathfrak{p}_0, \mathfrak{p}_1, \mathsf{ksK})$, *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathfrak{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$, *where*

$$\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot (\mathcal{B}_{\mathsf{acc}} + n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}).$$

*If* $\mathfrak{p}_0, \mathfrak{p}_1$ *are monomials, then the degree* $N$ *disappears from the above inequality.*

*Proof.* From correctness of $\mathsf{KeySwitch}$ we have that $\mathsf{acc}_{R,i} \in \mathsf{RLWE}_{\mathcal{B}_R,N,q}(\mathfrak{s}, \Delta_{q,t} \cdot \mathsf{m} \cdot \mathsf{L}^{i-1})$, where $\mathcal{B}_R \leq \mathcal{B}_{\mathsf{acc}} + n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}$. From correctness of $\mathsf{PubMux}$ we have that $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathfrak{s}, \Delta_{q,t} \cdot \mathfrak{p}_{\mathsf{m}})$, where $\mathcal{B}_{\mathsf{out}} \leq N \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_R$. Also from correctness of $\mathsf{PubMux}$ we have that if $\mathfrak{p}_0, \mathfrak{p}_1$ are monomials, then the degree $N$ disappears from the above inequalities. □

**Theorem 2.** *Let $f : \mathbb{Z}_t \mapsto \mathbb{Z}_{t'}$ be a function with a lookup table* $\mathsf{BootsMap}$ *such that for all $x \in \mathbb{Z}_t$, we have that* $\mathsf{BootsMap}[x+1] = \Delta_{Q,t'} f(x)$. *Let* $(\mathsf{rotP}_0, \mathsf{rotP}_1) \leftarrow \mathsf{SetupPolynomial}(\mathsf{BootsMap}, N)$ *for $N \in \mathbb{N}$. Let $y = \lfloor \frac{2 \cdot N}{t} \rceil \cdot \mathsf{m} + e$, for $\mathsf{m} \in \mathbb{Z}_t$ and $e \in [-\lceil \frac{N}{t} \rceil, \lceil \frac{N}{t} \rceil]$. Then*

- $\mathsf{Coefs}(\mathsf{rotP}_0 \cdot X^y)[1] = f(\mathsf{m})$ *for $y \in [0, N-1]$, and*
- $\mathsf{Coefs}(\mathsf{rotP}_1 \cdot X^y)[1] = f(\mathsf{m})$ *for $y \in [N, 2 \cdot N - 1]$.*

*Correctness of the Polynomial Construction.* Let us first consider the case $y = \lfloor \frac{2 \cdot N}{t} \rceil \cdot \mathsf{m} + e \in [0, N-1]$. In this case we have

$$\mathsf{rotP}_0 \cdot X^y = -\sum_{i=1}^{y} \mathbf{p}_0[N - y + i] \cdot X^{i-1} + \sum_{i=y+1}^{N} \mathbf{p}_0[i - y] \cdot X^{i-1} \in \mathcal{R}_{N,Q}$$

Hence, for $y = 0$, we have $\mathsf{Coefs}(\mathsf{rotP}_0 \cdot X^y)[1] = \mathbf{p}_0[1] = \mathsf{BootsMap}[\mathsf{m}+1] = \Delta_{Q,t'} f(\mathsf{m})$, and $\mathsf{Coefs}(\mathsf{rotP}_0 \cdot X^y)[1] = -\mathbf{p}_0[N - y + 1] = -(-\mathsf{BootsMap}[\mathsf{m}+1]) = \Delta_{Q,t'} f(\mathsf{m})$ for $y \in [1, N-1]$.

The case $y \in [N, 2 \cdot N - 1]$ is analogical. Let us first denote $y' + N = y$. In particular, we have

$$\begin{aligned} \mathsf{rotP}_1 \cdot X^y &= \mathsf{rotP}_1 \cdot X^{y'+N} = -\mathsf{rotP}_1 \cdot X^{y'} \\ &= -\Big( -\sum_{i=1}^{y'} \mathbf{p}_1[N - y' + i] \cdot X^{i-1} + \sum_{i=y'+1}^{N} \mathbf{p}_1[i - y'] \cdot X^{i-1} \Big) \in \mathcal{R}_{N,Q} \end{aligned}$$

Hence, for $y = N$, we have $y' = 0$ and $\mathsf{Coefs}(\mathsf{rotP}_1 \cdot X^y)[1] = -\mathbf{p}_1[1] = -(-\mathsf{BootsMap}[\mathsf{m} + 1]) = \Delta_{Q,t'} f(\mathsf{m})$. For $y \in (N, 2 \cdot N - 1]$ we have $y' \in (0, N-1]$ and $\mathsf{Coefs}(\mathsf{rotP}_1 \cdot X^y)[1] = \mathbf{p}_1[N - y' + 1] = \mathsf{BootsMap}[\mathsf{m}+1] = \Delta_{Q,t'} f(\mathsf{m})$. □

**Theorem 3** (Correctness of the Bootstrapping)**.** *Let $\mathbf{s} \in \mathbb{Z}_q^n$ for some $q, n \in \mathbb{N}$, $\mathfrak{s} \in \mathcal{R}_{N,q}$, and $\mathbf{s}_F \leftarrow \mathsf{KeyExt}(\mathfrak{s})$. Furthermore, let us define the following.*

- $\mathsf{brK} \leftarrow \mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$.
- $\mathsf{pK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{pK}}, N, 1, N, Q, \mathbf{s}_F, \mathfrak{s}, \mathsf{L}_{\mathsf{pK}})$.
- $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, N, n, 1, Q, \mathbf{s}_F, \mathbf{s}, \mathsf{L}_{\mathsf{ksK}})$.
- $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}}, n, q}(\mathbf{s}, \Delta_{q,t} \cdot \mathsf{m})$.
- $\mathsf{ct}_N \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q, 2 \cdot N)$, *and let us denote* $\mathsf{ct}_N \in \mathsf{LWE}_{\mathcal{B}_N, n, 2 \cdot N}(\mathbf{s}, \Delta_{2 \cdot N, t} \cdot \mathsf{m})$.

*Finally, let $\tilde{\mathsf{m}} \leftarrow \mathsf{Phase}(\mathsf{ct}_N)$, $\lfloor \tilde{\mathsf{m}} \rceil_t^{2 \cdot N} = \mathsf{m}$ and $\mathsf{rotP}_0, \mathsf{rotP}_1 \in \mathcal{R}_{N,Q}$ be such that*

- *if $\tilde{\mathsf{m}} \in [0, N)$, then* $\mathsf{Coefs}(X^{\tilde{\mathsf{m}}} \cdot \mathsf{rotP}_0)[1] = \Delta_{Q,t'} \cdot f(\mathsf{m})$, *and*
- *if $\tilde{\mathsf{m}} \in [N, 2 \cdot N)$, then* $\mathsf{Coefs}(X^{\tilde{\mathsf{m}}} \cdot \mathsf{rotP}_1)[1] = \Delta_{Q,t'} \cdot f(\mathsf{m})$, *where $f : \mathbb{Z}_t \mapsto \mathbb{Z}_{t'}$.*

*If* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}_0, \mathsf{rotP}_1, \mathsf{L}_{\mathsf{boot}}, \mathsf{pK}, \mathsf{ksK})$ *and given that $\mathcal{B}_N \leq \frac{N}{t}$, then* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}}, n, q}(\mathbf{s}, \Delta_{q,t'} \cdot f(\mathsf{m}))$, *where*

$$\mathcal{B}_N \leq \Big( \frac{2 \cdot N}{q} \Big)^2 \cdot \mathcal{B}_{\mathsf{ct}} + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty,$$

*for $N \neq 2 \cdot q$, and $\mathcal{B}_N = \mathcal{B}_{\mathsf{ct}}$ otherwise, and $\mathcal{B}_{\mathsf{ct}} \leq \max(\mathcal{B}_{\mathsf{fresh}}, \mathcal{B}_{\mathsf{out}})$. We use the bound $\mathcal{B}_{\mathsf{fresh}}$ if* $\mathsf{ct}$ *is a freshly encrypted ciphertext. Finally the bound $\mathcal{B}_{\mathsf{out}}$ of the bootstrapped ciphertext* $\mathsf{ct}_{\mathsf{out}}$ *that is as follows*

$$\mathcal{B}_{\mathsf{out}} \leq \frac{q^2}{Q^2} \cdot (\mathcal{B}_F + \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}) + \mathsf{Ham}(\mathbf{s}_F) \cdot ||\mathsf{Var}(\mathbf{s}_F)||_\infty,$$

*for $Q \neq q$, and $\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_F + \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}$ for $Q = q$, where*

- $\mathcal{B}_F \leq N \cdot \ell_{\mathsf{boot}} \cdot \mathsf{L}^2_{\mathsf{boot}} \cdot \big(\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_P\big)$
- $\mathcal{B}_{\mathsf{BR}} \leq 2 \cdot n \cdot u \cdot N \cdot \ell_{\mathsf{RGSW}} \cdot \mathsf{L}^2_{\mathsf{RGSW}} \cdot \mathcal{B}_{\mathsf{brK}}$,
- $\mathcal{B}_{KS} \leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}^2_{\mathsf{ksK}} \cdot \mathcal{B}_{\mathsf{ksK}}$, *and*
- $\mathcal{B}_P \leq N \cdot \ell_{\mathsf{pK}} \cdot \mathsf{L}^2_{\mathsf{pK}} \cdot \mathcal{B}_{\mathsf{pK}}$.

*Proof.* Let us denote $\mathsf{Phase}\big(\mathsf{ModSwitch}(\mathsf{ct}, q, 2N)\big) = \tilde{\mathsf{m}} = \Delta_{2\cdot N,t} \cdot \mathsf{m} + e \mod 2N$. From correctness of modulus switching we have $||\mathsf{Var}(e)||_\infty \leq \mathcal{B}_N \leq \left(\frac{2\cdot N}{q}\right)^2 \cdot \mathcal{B}_{\mathsf{ct}} + \mathsf{Ham}(\mathbf{s}) \cdot ||\mathsf{Var}(\mathbf{s})||_\infty$. The first $\ell_{\mathsf{boot}}$ blind rotations compute powers of $\mathsf{L}_{\mathsf{boot}}$ of the sign function. Specifically from the correctness of blind rotation we have $\mathsf{acc}_{\mathsf{BR},i} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{BR}},N,Q}(\mathfrak{s}, \mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2} \cdot \mathsf{sgnP} \cdot X^{\tilde{\mathsf{m}}})$. Hence, if $\tilde{\mathsf{m}} \mod 2N \in [0, N)$, then the constant coefficient is equal to $\mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2} \in \mathbb{Z}_Q$. If $\tilde{\mathsf{m}} \mod 2N \in [N, 2 \cdot N)$ then the constant coefficient is equal to $-\mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2} \in \mathbb{Z}_Q$. Equivalently, the constant coefficient is $\mathsf{sgn}(\mathsf{m}) \cdot \mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2} \in \mathbb{Z}_Q$. Note that $\mathsf{sgn}(\tilde{\mathsf{m}}) \mod 2 \cdot N = \mathsf{sgn}(\mathsf{m}) \mod t$. From correctness of sample extraction and additive homomorphism of LWE samples we have $\mathsf{acc}_{\mathbf{c},i} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{BR}},N,Q}(\mathbf{s}_{\mathsf{sgn}}, \mathsf{m}_{\mathsf{BR},i})$, where $\mathcal{B}_{\mathsf{BR}}$ is the error stemming from the blind rotation, and $\mathsf{m}_{\mathsf{BR},i} = \mathsf{sgn}(\mathsf{m}) \cdot \mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2} + \mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \frac{\Delta_{Q,t}}{2}$. Note that if $\mathsf{sgn}(\mathsf{m}) = 1$, then $\mathsf{m}_{\mathsf{BR},i} = \mathsf{L}^{i-1}_{\mathsf{boot}} \cdot \Delta_{Q,t}$, and if $\mathsf{sgn}(\mathsf{m}) = -1$, then $\mathsf{m}_{\mathsf{BR},i} = 0$.

The next step is to create a new accumulator holding the polynomial $\mathsf{rotP}_{\mathsf{inter}} \in \mathcal{R}_{N,Q}$, where $\mathsf{inter} = 0$ if $\mathsf{m} + e \in [0, N)$ and $\mathsf{inter} = 1$ if $\mathsf{m} + e \in [N, 2N)$. From correctness of the accumulator builder we have the $\mathsf{acc}_F \in \mathsf{RLWE}_{\mathcal{B}_F,N,Q}(\mathfrak{s}, \mathsf{rotP}_{\mathsf{inter}})$, where

$$\mathcal{B}_F \leq N \cdot \ell_{\mathsf{boot}} \cdot \mathsf{L}^2_{\mathsf{boot}} \cdot \big(\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_P\big)$$

with $\mathcal{B}_P$ being the bound on the error induced by the LWE to RLWE key switching algorithm.

From correctness of blind rotation we have that $\mathsf{acc}_{\mathsf{BR},F} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{BR},F},N,Q}(\mathfrak{s}, \mathsf{rotP}_{\mathsf{inter}} \cdot X^{\tilde{\mathsf{m}} \mod 2N})$. Hence, from correctness of sample extraction and given that $\Delta_{Q,t'} \cdot f(\mathsf{m}) = \mathsf{Coefs}(\mathsf{rotP}_{\mathsf{inter}} \cdot X^{\tilde{\mathsf{m}} \mod 2N})[1]$, we have $\mathbf{c}_Q \in \mathsf{LWE}_{\mathcal{B}_{\mathbf{c},Q},N,Q}(\mathbf{s}_F,\ \Delta_{Q,t'} \cdot f(\mathsf{m}))$, where $\mathcal{B}_{\mathbf{c},Q} \leq \mathcal{B}_F + \mathcal{B}_{\mathsf{BR}}$.

From correctness of key switching we have that $\mathbf{c}_{Q,\mathsf{ksK}} \in \mathsf{LWE}_{\mathcal{B},n,q}(\mathbf{s}, \Delta_{Q,t'} \cdot f(\mathsf{m}))$, where $\mathcal{B}_{\mathbf{c},Q,\mathsf{ksK}} \leq \mathcal{B}_{\mathbf{c},Q} + \mathcal{B}_{KS}$, where $\mathcal{B}_{KS}$ is the bound induced by the key switching procedure. From correctness of modulus switching we have that $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}},n,q}(\mathbf{s}, \Delta_{q,t'} \cdot f(\mathsf{m}))$, where

$$\mathcal{B}_{\mathsf{out}} \leq \Big(\frac{q^2}{Q^2}\Big) \cdot (\mathcal{B}_F + \mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}) + \mathsf{Ham}(\mathbf{s}_F) \cdot ||\mathsf{Var}(\mathbf{s}_F)||_\infty.$$

Recall that the errors induced by the blind rotation is $\mathcal{B}_{\mathsf{BR}} \leq 2 \cdot n \cdot u \cdot N \cdot \ell_{\mathsf{RGSW}} \cdot \mathsf{L}^2_{\mathsf{RGSW}} \cdot \mathcal{B}_{\mathsf{brK}}$. Similarly, recall that the error induced by the LWE to RLWE algorithm is $\mathcal{B}_P \leq N \cdot \ell_{\mathsf{pK}} \cdot \mathsf{L}^2_{\mathsf{pK}} \cdot \mathcal{B}_{\mathsf{pK}}$. For LWE to LWE key switching the error is $\mathcal{B}_{KS} \leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}^2_{\mathsf{ksK}} \cdot \mathcal{B}_{\mathsf{ksK}}$. □

# B Specifications of Algorithms from the Preliminaries

**Definition 14** (External Product)**.** The external product $\mathsf{extProd}$ given as input $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$ and $\mathbf{d} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})$, outputs the following

$$\mathsf{extProd}(\mathbf{C}, \mathbf{d}) = \mathsf{G}^{-1}_{\mathsf{L},q}\big(\mathbf{d}^\top\big) \cdot \mathbf{C}.$$

| $\mathsf{Mux}(\mathbf{C}, \mathbf{g}, \mathbf{h})$ |
|---|
| **Input:** Takes as input $\mathbf{C} \in \mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}})$, where $\mathsf{m}_{\mathbf{C}} \in \mathbb{B}$, $\mathbf{g} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{g}},n,N,q}(\mathbf{s}, .)$ and $\mathbf{h} \in \mathsf{GLWE}_{\mathcal{B}_{\mathbf{h}},n,N,q}(\mathbf{s}, .)$. **Output:** A ciphertext $\mathbf{c}_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n,N,q}(\mathbf{s}, \mathsf{m}_{\mathsf{out}})$ where $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{h}}$ **for** $\mathsf{m}_{\mathbf{C}} = 0$ and $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{g}}$ **for** $\mathsf{m}_{\mathbf{C}} = 1$. |
| 1 : Compute $\mathbf{d} \leftarrow \mathbf{g} - \mathbf{h}$. |
| 2 : Compute $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{extProd}(\mathbf{C}, \mathbf{d})^\top + \mathbf{h}$. |
| 3 : Output $\mathbf{c}_{\mathsf{out}}$. |

**Figure 5:** CMux gate.

| $\mathsf{ModSwitch}(\mathbf{c}, Q, q)$ |
|---|
| **Input:** Takes as input $\mathbf{c} = [b, \mathbf{a}^\top]^\top \in \mathsf{LWE}_{\mathcal{B},n,Q}(\mathbf{s}, .)$, $Q \in \mathbb{N}$ and $q \in \mathbb{N}$. **Output:** A ciphertext $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B},n,q}\left(\mathbf{s}, \mathsf{m} \cdot \left(\frac{q}{t}\right)\right)$. |
| 1 : Set $b_q \leftarrow \lfloor \frac{q \cdot b}{Q} \rceil$. |
| 2 : Set $\mathbf{a}_q = \lfloor \frac{q \cdot \mathbf{a}}{Q} \rceil$. |
| 3 : Output $\mathsf{ct}_{\mathsf{out}} \leftarrow [b_q, \mathbf{a}_q^\top]^\top \in \mathbb{Z}_q^{(n+1)\times 1}$. |

**Figure 6:** Modulus switching algorithm.

To conveniently describe our $\mathsf{SampleExt}$, we first need to introduce a step function which we call $\mathsf{NSgn} : \mathbb{R} \mapsto \{-1, 1\}$ and which is defined as

$$\mathsf{NSgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Note that the function is equal in all points to the sign function except for $x = 0$, which its image is $-1$.

# C Correctness Analysis for Preliminaries

*of lemma 1 - Linear Homomorphism of GLWE samples.* Denote $\mathbf{c} = [\mathfrak{b}_{\mathbf{c}}, \mathbf{a}_{\mathbf{c}}^\top]^\top$, where $\mathfrak{b}_{\mathbf{c}} = \mathbf{a}_{\mathbf{c}}^\top \cdot \mathbf{s} + \mathsf{m}_{\mathbf{c}} + \mathfrak{e}_{\mathbf{c}}$ and $\mathbf{d} = [\mathfrak{b}_{\mathbf{d}}, \mathbf{a}_{\mathbf{d}}^\top]^\top$, where $\mathfrak{b}_{\mathbf{d}} = \mathbf{a}_{\mathbf{d}}^\top \cdot \mathbf{s} + \mathsf{m}_{\mathbf{d}} + \mathfrak{e}_{\mathbf{d}}$. Then we have $\mathbf{c}_{\mathsf{out}} = [\mathfrak{b}, \mathbf{a}^\top]^\top = \mathbf{c} + \mathbf{d} = [\mathfrak{b}_{\mathbf{c}} + \mathfrak{b}_{\mathbf{d}}, (\mathbf{a}_{\mathbf{c}} + \mathbf{a}_{\mathbf{d}})^\top]^\top$, and $\mathfrak{b} = \mathbf{a}^\top \cdot \mathbf{s} + \mathsf{m} + \mathfrak{e}$, where $\mathsf{m} = \mathsf{m}_{\mathbf{c}} + \mathsf{m}_{\mathbf{d}}$ and $\mathfrak{e} = \mathfrak{e}_{\mathbf{c}} + \mathfrak{e}_{\mathbf{d}}$. Thus, we have that $\mathsf{Error}(\mathbf{c}_{\mathsf{out}}) = \mathfrak{e}_{\mathbf{c}} + \mathfrak{e}_{\mathbf{d}}$, and $||\mathsf{Var}(\mathsf{Error}(\mathbf{c}_{\mathsf{out}}))||_\infty \leq \mathcal{B}_{\mathbf{c}} + \mathcal{B}_{\mathbf{d}}$.

Let $\mathfrak{d} \in \mathcal{R}_{N,\mathsf{L}}$ and $\mathbf{c}_{\mathsf{out}} = \mathbf{c} \cdot \mathfrak{d} = [\mathfrak{b}_{\mathfrak{d}}, \mathbf{a}_{\mathfrak{d}}^\top]^\top$, where $\mathbf{a}_{\mathfrak{d}} = \mathbf{a} \cdot \mathfrak{d}$ and $\mathfrak{b}_{\mathfrak{d}} = \mathbf{a}_{\mathfrak{d}}^\top \cdot \mathbf{s} + \mathsf{m}_{\mathbf{c}} \cdot \mathfrak{d} + \mathfrak{e}_{\mathbf{c}} \cdot \mathfrak{d}$.

If $\mathfrak{d}$ is a monomial with $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$, then we have $||\mathsf{Var}(\mathsf{Error}(\mathbf{c}_{\mathsf{out}}))||_\infty \leq ||\mathcal{B}_{\mathfrak{d}}^2 \cdot \mathsf{Var}(\mathfrak{e}_{\mathbf{c}})||_\infty \leq \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$.

If $\mathfrak{d} \in \mathcal{R}_{N,\mathsf{L}}$ is a polynomial with $||\mathfrak{d}||_\infty \leq \mathcal{B}_{\mathfrak{d}}$ then $||\mathsf{Var}(\mathsf{Error}(\mathfrak{d} \cdot \mathbf{c}))||_\infty \leq N \cdot \mathcal{B}_{\mathfrak{d}}^2 \cdot \mathcal{B}_{\mathbf{c}}$. □

*of lemma 2 - Linear Homomorphism of GGSW samples.* The proof follows from the fact that rows of GGSW samples are GLWE samples. □

*of lemma 3 - Correctness of the External Product.* Denote $\mathbf{C} = \mathbf{A} + \mathsf{m}_{\mathbf{C}} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)} \in \mathcal{R}_{N,q}^{(n+1)\ell\times(n+1)}$ and $\mathbf{d} = [\mathfrak{b}, \mathbf{a}^\top]^\top \in \mathcal{R}_{N,q}^{(n+1)\times 1}$. To ease the exposition, denote $\mathbf{r} = \mathsf{G}_{\mathsf{L},q}^{-1}(\mathbf{d}^t) \in$

$\mathsf{SampleExt}(\mathsf{ct}, k)$ :

**Input:** Takes as input $\mathsf{ct} = [\mathfrak{b}, \mathfrak{a}]^\top \in \mathsf{RLWE}_{\mathcal{B},N,q}(\mathfrak{s}, .)$
and an index $k \in [N]$.

**Output:** A ciphertext $\mathbf{c} \in \mathsf{LWE}_{\mathcal{B},N,q}(\mathbf{s}, \mathsf{m}_k)$.

Set $b' = \mathsf{Coefs}(\mathfrak{b})[k]$.
Set $\mathbf{a} \leftarrow \mathsf{Coefs}(\mathfrak{a}) \in \mathbb{Z}_q^N$.
For all $i \in [N]$
  Set $\mathbf{a}'[i] \leftarrow \mathsf{NSgn}(k - i + 1) \cdot \mathbf{a}[(k - i \bmod N) + 1]$.
Return $\mathbf{c} \leftarrow (\mathbf{a}', b')$.

**Figure 7:** Sample and key extraction. The $\mathsf{KeyExt}(\mathfrak{s})$ function on input key $\mathfrak{s} \in \mathcal{R}_{N,q}$ outputs its coefficient vector. That is it outputs $\mathsf{Coefs}(\mathfrak{s})$.

$\mathsf{KeySwitchSetup}(\mathcal{B}_{\mathsf{ksK}}, n, n', N, q, \mathbf{s}, \mathbf{s}', \mathsf{L}_{\mathsf{ksK}})$

**Input:** A bound $\mathcal{B}_{\mathsf{ksK}} \in \mathbb{N}$, dimensions $n, n' \in \mathbb{N}$, a degree $N \in \mathbb{N}$ and a modulus $q \in \mathbb{N}$ that defines $\mathcal{R}_{N,q}$, vectors $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{s}' \in \mathcal{R}_{N,q}^{n'}$ and a basis $\mathsf{L}_{\mathsf{ksK}} \in \mathbb{N}$.

**Output:** A key switching key $\mathsf{ksK} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', .)^{n \times \ell_{\mathsf{ksK}}}$.

Set $\ell_{\mathsf{ksK}} = \lceil \log_{\mathsf{L}_{\mathsf{ksK}}} q \rceil$.
For $i \in [n]$, $j \in [\ell_{\mathsf{ksK}}]$
  $\mathsf{ksK}[i,j] \leftarrow \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}\left(\mathbf{s}', \mathbf{s}[i] \cdot \mathsf{L}_{\mathsf{ksK}}^{j-1}\right)$.
Output $\mathsf{ksK} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', .)^{n \times \ell_{\mathsf{ksK}}}$.

**Figure 8:** Key switching setup

$\mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK})$

**Input:** Takes as input $\mathbf{c} = [b, \mathbf{a}^\top]^\top \in \mathsf{LWE}_{\mathcal{B}_{\mathbf{c}},n,q}(\mathbf{s}, .)$
and a key switching key $\mathsf{ksK} \in \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', .)^{n \times \ell_{\mathsf{ksK}}}$.

**Output:** A ciphertext $c_{\mathsf{out}} \in \mathsf{GLWE}_{\mathcal{B},n',N,q}(\mathbf{s}', \mathsf{m})$.

Denote $\ell_{\mathsf{ksK}} = \lceil \log_{\mathsf{L}_{\mathsf{ksK}}} q \rceil$.
Compute $\mathbf{A} \leftarrow \mathsf{G}^{-1}_{\mathsf{L}_{\mathsf{ksK}},q}(\mathbf{a}) \in \mathbb{Z}_{\mathsf{L}_{\mathsf{ksK}}}^{n \times \ell_{\mathsf{ksK}}}$.
Output $c_{\mathsf{out}} \leftarrow [\mathfrak{b}, 0]^\top - \sum_{i=1}^{n} \sum_{j=1}^{\ell_{\mathsf{ksK}}}, \mathbf{A}[i,j] \cdot \mathsf{ksK}[i,j]$.

**Figure 9:** Key switching algorithm and its setup.

$\mathsf{BRKeyGen}(n, \mathbf{s}, \mathbf{u}, \mathcal{B}_{\mathsf{brK}}, N, Q, \mathsf{L}_{\mathsf{RGSW}}, \mathfrak{s})$

**Input:** A dimension $n \in \mathbb{N}$, a secret key $\mathbf{s} \in \mathbb{Z}_t^{n \times 1}$,
a vector $\mathbf{u} \in \mathbb{Z}^u$, a bound $\mathcal{B}_{\mathsf{brK}}$, a degree $N \in \mathbb{N}$,
a modulus $Q$ defining the ring $\mathcal{R}_{N,Q}$, a basis $\mathsf{L}_{\mathsf{RGSW}} \in \mathbb{N}$,
and a secret key $\mathfrak{s} \in \mathcal{R}_{N,Q}$.
**Output:** A blind rotation key $\mathsf{brK} \in \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{RGSW}}}(\mathfrak{s}, .)^{n \times u}$.

Set a matrix $\mathbf{Y} \in \mathbb{B}^{n \times u}$ as follows:
For $i \in [n]$ do
  Set the $i$-th row of $\mathbf{Y}$, **to** satisfy $\mathbf{s}[i] = \sum_{j=1}^{u} \mathbf{Y}[i,j] \cdot \mathbf{u}[j]$,
For $i \in [n]$, $j \in [u]$
  Set $\mathsf{brK}[i,j] = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{RGSW}}}(\mathfrak{s}, \mathbf{Y}[i,j])$.
Output $\mathsf{brK}$.

**Figure 10:** Blind rotation setup

$\mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}, \mathsf{ct}, \mathbf{u})$

**Input:** Blind rotation key $\mathsf{brK} = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{RGSW}}}(\mathfrak{s}, .)^{n \times u}$,
an accumulator $\mathsf{acc} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{acc}},N,Q}(\mathfrak{s}, .)$,
a ciphertext $\mathsf{ct} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,2\cdot N}(\mathbf{s}, .)$,
where $\mathbf{s} \in \mathbb{Z}_t^n$ with $t \leq q$, and a vector $\mathbf{u} \in \mathbb{Z}^u$.
**Output:** A ciphertext $\mathsf{acc}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,Q}(\mathfrak{s}, \mathsf{m}_{\mathsf{acc}}^{\mathsf{out}})$.

Let $\mathbf{c} = [b, \mathbf{a}^\top]^\top \in \mathbb{Z}_{2\cdot N}^{(n+1)\times 1}$.
For $i \in [n]$ do
  For $j \in [u]$ do
    $\mathsf{acc} \leftarrow \mathsf{Mux}\big(\mathsf{brK}[i,j], \mathsf{acc} \cdot X^{-\mathbf{a}[i]\cdot\mathbf{u}[j]}, \mathsf{acc}\big)$.
Output $\mathsf{acc}$.

**Figure 11:** Blind Rotation.

$\mathsf{Bootstrap}(\mathsf{brK}, \mathbf{u}, \mathsf{ct}, \mathsf{rotP}, \mathsf{ksK})$

**Input:** Blind rotation key $\mathsf{brK} = \mathsf{RGSW}_{\mathcal{B}_{\mathsf{brK}},N,Q,\mathsf{L}_{\mathsf{brK}}}(\mathfrak{s}, .)^{n \times u}$,
a vector $\mathbf{u} \in \mathbb{Z}^u$,
a LWE sample $\mathsf{ct} = \mathsf{LWE}_{\mathcal{B}_{\mathsf{ct}},n,q}(\mathbf{s}, .) = [b, \mathbf{a}^\top]^\top \in \mathbb{Z}_q^{(n+1)\times 1}$,
a polynomial $\mathsf{rotP} \in \mathcal{R}_{N,Q}$,
and a LWE **to** LWE keyswitch key $\mathsf{ksK}$.
**Output:** A ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathbf{s}, \mathsf{m}_{\mathsf{out}})$.

Set $\mathsf{ct}_N \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q, 2 \cdot N) = [b_N, \mathbf{a}_N^\top]^\top$.
Set $\mathsf{rotP}' \leftarrow \mathsf{rotP} \cdot X^{b_N} \in \mathcal{R}_{N,Q}$.
Run $\mathsf{acc}_F \leftarrow [\mathsf{rotP}', 0]^\top$
Run $\mathsf{acc}_{\mathsf{BR},F} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_F, \mathsf{ct}_N, \mathbf{u})$.
Run $\mathbf{c}_Q \leftarrow \mathsf{SampleExt}(\mathsf{acc}_{\mathsf{BR},F}, 1)$.
Run $\mathbf{c}_{Q,\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}_Q, \mathsf{ksK})$.
Run $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{Q,\mathsf{ksK}}, Q, q)$.
Return $\mathsf{ct}_{\mathsf{out}}$.

**Figure 12:** TFHE Bootstrapping.

$\mathcal{R}_{\mathsf{L}}^{1\times(n+1)\ell}$ and $\mathbf{A} = [\mathbf{c}_1, \ldots, \mathbf{c}_{(n+1)\ell}]^\top$, where for $i \in [(n+1)\ell]$ the vector $\mathbf{c}_i$ is the transposed GLWE sample of zero making up the row of the matrix $\mathbf{A}$. Then from the definition of external product we have $\mathbf{r} \cdot \mathbf{C} = \mathbf{r} \cdot \mathbf{A} + \mathsf{m}_{\mathbf{C}} \cdot \mathbf{r} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)}$. From correctness of the gadget decomposition, we have that $\mathbf{r} \cdot \mathbf{G}_{\ell,\mathsf{L},(n+1)} = \mathbf{d} \in \mathcal{R}_{N,q}^{(n+1)\times 1}$. Then, $\mathbf{c}' = \mathbf{r} \cdot \mathbf{A} = \sum_{i=1}^{(n+1)\ell} \mathbf{r}[i] \cdot \mathbf{c}_i \in \mathcal{R}_{N,q}^{(n+1)\times 1}$ is a transposed GLWE sample of zero. Thus we can write $\mathbf{c} = \sum_{i=1}^{(n+1)\ell} \mathbf{r}[i] \cdot \mathbf{c}_i + \mathsf{m}_{\mathbf{C}} \cdot \mathbf{d}$ and we have $\mathbf{c}^\top$ is a valid GLWE sample of $\mathsf{m}_{\mathbf{C}} \cdot \mathsf{m}_{\mathbf{d}}$. Finally $\mathcal{B} = ||\mathsf{Var}(\mathsf{Error}(c))||_\infty$ follows from the analysis of linear combinations of GLWE samples. In particular, we have

- $\mathcal{B} \leq N \cdot (n+1) \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{C}} + N \cdot \mathcal{B}_{\mathsf{m}_{\mathbf{C}}}^2 \cdot \mathcal{B}_{\mathbf{d}}$ in general, and
- $\mathcal{B} \leq N \cdot (n+1) \cdot \ell \cdot \mathsf{L}^2 \cdot \mathcal{B}_{\mathbf{C}} + \mathcal{B}_{\mathsf{m}_{\mathbf{C}}}^2 \cdot \mathcal{B}_{\mathbf{d}}$ when $\mathsf{m}_{\mathbf{C}}$ is a monomial.

□

*of lemma 4 - Correctness of the* $\mathsf{Mux}$ *Gate.* Let us first denote $\mathbf{d} = \mathbf{g} - \mathbf{h} = \mathsf{GLWE}_{\mathcal{B}_{\mathbf{g}}}(\mathbf{s}, \mathsf{m}_{\mathbf{g}}) - \mathsf{GLWE}_{\mathcal{B}_{\mathbf{h}}}(\mathbf{s}, \mathsf{m}_{\mathbf{h}}) = \mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}}}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})$, where $\mathsf{m}_{\mathbf{d}} = \mathsf{m}_{\mathbf{g}} - \mathsf{m}_{\mathbf{h}}$. Then from correctness of the external product and linear homomorphism of GLWE samples we have

$$\begin{aligned}
\mathbf{c}_{\mathsf{out}} &= \mathsf{extProd}(\mathbf{C}, \mathbf{d}) + \mathbf{h} \\
&= \mathsf{extProd}(\mathsf{GGSW}_{\mathcal{B}_{\mathbf{C}},n,N,q,\mathsf{L}}(\mathbf{s}, \mathsf{m}_{\mathbf{C}}), +\mathsf{GLWE}_{\mathcal{B}_{\mathbf{d}}}(\mathbf{s}, \mathsf{m}_{\mathbf{d}})) + \mathsf{GLWE}_{\mathcal{B}_{\mathbf{h}}}(\mathbf{s}, \mathsf{m}_{\mathbf{h}}) \\
&= \mathsf{GLWE}_{\mathcal{B}}(\mathbf{s}, \mathsf{m}_{\mathsf{out}}),
\end{aligned}$$

where $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{C}} \cdot (\mathsf{m}_{\mathbf{g}} - \mathsf{m}_{\mathbf{h}}) + \mathsf{m}_{\mathbf{h}}$. Hence if $\mathsf{m}_{\mathsf{out}} = 0$, we have $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{h}}$, and if $\mathsf{m}_{\mathsf{out}} = 1$, we have $\mathsf{m}_{\mathsf{out}} = \mathsf{m}_{\mathbf{g}}$.

For the error analysis let us denote explicitly $\mathsf{Error}(\mathbf{g}) = \mathfrak{e}_{\mathbf{g}}$ and $\mathsf{Error}(\mathbf{h}) = \mathfrak{e}_{\mathbf{h}}$. Thus we also have $\mathsf{Error}(\mathbf{d}) = \mathfrak{e}_{\mathbf{d}} = \mathfrak{e}_{\mathbf{g}} = \mathfrak{e}_{\mathbf{h}}$. We also denote $\mathsf{Error}(\mathbf{C}[i]^\top) = \mathfrak{e}_i$ for $i \in [(n+1) \cdot \ell]$ and $\ell = \lceil \log_{\mathsf{L}} q \rceil$, and $\mathsf{G}_{\mathsf{L},q}^{-1}(\mathbf{d}) = \mathbf{r}$. Then, from the correctness analysis of the external product we have that

$$\begin{aligned}
\mathsf{Error}(\mathbf{c}_{\mathsf{out}}) &= \sum_{i=1}^{(n+1)\cdot\ell} \mathbf{r}[i]\cdot\mathfrak{e}_i + \mathsf{m}_{\mathbf{C}}\cdot\mathfrak{e}_{\mathbf{d}} + \mathfrak{e}_{\mathbf{h}} \\
&= \sum_{i=1}^{(n+1)\cdot\ell} \mathbf{r}[i]\cdot\mathfrak{e}_i + \mathsf{m}_{\mathbf{C}}\cdot\mathfrak{e}_{\mathbf{g}} - \mathsf{m}_{\mathbf{C}}\cdot\mathfrak{e}_{\mathbf{h}} + \mathfrak{e}_{\mathbf{h}}.
\end{aligned}$$

Thus if we have $\mathsf{m}_{\mathbf{C}} = 0$, then $\mathsf{Error}(\mathbf{c}_{\mathsf{out}}) = \sum_{i=1}^{(n+1)\cdot\ell} \mathbf{r}[i]\cdot\mathfrak{e}_i + \mathfrak{e}_{\mathbf{h}}$, and if $\mathsf{m}_{\mathbf{C}} = 1$, we obtain $\mathsf{Error}(\mathbf{c}_{\mathsf{out}}) = \sum_{i=1}^{(n+1)\cdot\ell} \mathbf{r}[i]\cdot\mathfrak{e}_i + \mathfrak{e}_{\mathbf{g}}$.

Finally, we can place an upper on the variance $\mathcal{B} \leq (n+1)\cdot N\cdot\ell\cdot\mathsf{L}^2\cdot\mathcal{B}_{\mathbf{C}} + \max(\mathcal{B}_{\mathbf{d}}, \mathcal{B}_{\mathbf{h}})$. □

*of lemma 5 - Correctness of Modulus Switching.* Denote $\mathbf{c} = (b, \mathbf{a})$, where $b = \mathbf{a}^\top\cdot\mathbf{s} + \Delta_{Q,t}\cdot\mathsf{m} + e \in \mathbb{Z}_Q$ and $||\mathsf{Var}(e)||_\infty \leq \mathcal{B}_{\mathbf{c}}$. From the assumption that $Q = 0 \mod t$ we have $\Delta_{Q,t} = \lfloor \frac{Q}{t} \rceil = \frac{Q}{t}$. Then we have that the following:

$$\begin{aligned}
\mathsf{Phase}\Big(\lfloor \frac{q}{Q}\cdot\mathbf{c} \rceil\Big) &= \lfloor \frac{q}{Q}\cdot b \rceil - \lfloor \frac{q}{Q}\cdot\mathbf{a}^\top \rceil\cdot\mathbf{s} \\
&= \frac{q}{Q}\cdot b + r - \frac{q}{Q}\cdot\mathbf{a}^\top\cdot\mathbf{s} + \mathbf{r}^\top\cdot\mathbf{s} \\
&= \frac{q}{Q}\cdot\Delta_{Q,t}\cdot\mathsf{m} + \frac{q}{Q}\cdot e + r + \mathbf{r}^\top\cdot\mathbf{s} \\
&= \frac{q}{t}\cdot\mathsf{m} + \frac{q}{Q}\cdot e + r + \mathbf{r}^\top\cdot\mathbf{s}
\end{aligned}$$

where $r \in \mathbb{R}$ and $\mathbf{r} \in \mathbb{R}^n$ are in $[-\frac{1}{2}, \frac{1}{2}]$. Furthermore, $r$ and $\mathbf{r}$ are close to uniform distribution over their support. Let us denote $\mathsf{Ham}(\mathbf{s}) = h$. Therefore, we have

$$\begin{aligned}
&\Big|\Big|\mathsf{Var}\Big(\mathsf{Error}(\lfloor \frac{q}{Q}\cdot\mathbf{c} \rceil)\Big)\Big|\Big|_\infty \\
&= \Big|\Big|\mathsf{Var}\Big(\frac{q}{Q}\cdot e + r + \mathbf{r}^\top\cdot\mathbf{s}\Big)\Big|\Big|_\infty \\
&= \Big|\Big|\mathsf{Var}(\frac{q}{Q}\cdot e)\Big|\Big|_\infty + \Big|\Big|\mathsf{Var}(\mathbf{r}^\top\cdot\mathbf{s})\Big|\Big|_\infty \\
&\leq \frac{q^2}{Q^2}\cdot\mathcal{B}_{\mathbf{c}} + \sum_{i=1}^{\mathsf{Ham}(\mathbf{s})} ||\mathsf{Var}(\mathbf{r}[i]\cdot\mathbf{s}[i])||_\infty \\
&\leq \frac{q^2}{Q^2}\cdot\mathcal{B}_{\mathbf{c}} + \mathsf{Ham}(\mathbf{s})\cdot||\mathsf{Var}(\mathbf{s})||_\infty
\end{aligned}$$

In the above we assume that $||\mathsf{Var}(\mathbf{s}[i])||_\infty = ||\mathsf{Var}(\mathbf{s})||_\infty$ for all $i \in [n]$.

The expectation of the output noise satisfies

$$\begin{aligned}
&\Big|\frac{q}{Q}\cdot\mathsf{E}\big(\mathsf{Error}(\mathbf{c})\big) + \mathsf{E}\big(\frac{q}{Q}\cdot e + r + \mathbf{r}^\top\cdot\mathbf{s}\big)\Big| \\
&= \big|\frac{q}{Q}\cdot\mathsf{E}(\mathsf{Error}(\mathbf{c}))\big| + \big|\mathsf{E}(r + \mathbf{r}^\top\cdot\mathbf{s})\big| \\
&= \big|\frac{q}{Q}\cdot\mathsf{E}(\mathsf{Error}(\mathbf{c}))\big| + \big|\mathsf{E}(r) + \sum_{i=1}^{n}\mathsf{E}(\mathbf{r}\cdot\mathbf{s})\big| \\
&\leq \big|\frac{q}{Q}\cdot\mathsf{E}(\mathsf{Error}(\mathbf{c}))\big| + 1/2 + 1/2\cdot\mathsf{Ham}(\mathbf{s})\cdot\big|\mathsf{E}(\mathbf{s})\big|
\end{aligned}$$

given that the expectation of $e$ is 0.

□

*of lemma 6 - Correctness of Sample Extraction.* Denote $\mathbf{s} = \mathsf{Coefs}(\mathfrak{s}) \in \mathbb{Z}_q^N$ and $b' = \mathsf{Coefs}(\mathfrak{b})[k] \in \mathbb{Z}_q$ and $\mathbf{a} = \mathsf{Coefs}(\mathfrak{a}) \in \mathbb{Z}_q^N$. Denote $\mathfrak{b} = \mathfrak{a} \cdot \mathfrak{s} + \mathsf{m} + \mathfrak{e} \in \mathcal{R}_{N,q}$, $\mathsf{m} = \sum_{i=1}^N \mathsf{m}_i \cdot X^{i-1}$ and $\mathfrak{e} = \sum_{i=1}^N e_i \cdot X^{i-1}$ then it is easy to see, that $b' = \mathsf{Coefs}(\mathfrak{a} \cdot \mathfrak{s})[k] + \mathsf{Coefs}(\mathsf{m})[k] + \mathsf{Coefs}(\mathfrak{e}) = \mathsf{Coefs}(\mathfrak{a} \cdot \mathfrak{s})[k] + \mathsf{m}_k + e_k$. Furthermore, denote $\mathfrak{a} = \sum_{i=1}^N a_i \cdot X^{i-1}$ and $\mathfrak{s} = \sum_{i=1}^N s_i \cdot X^{i-1}$. Denote $\mathfrak{s} \cdot \mathfrak{a} = \big(\sum_{i=1}^N a_i \cdot X^{i-1}\big) \cdot \big(\sum_{i=1}^N s_i \cdot X^{i-1}\big)$. By expanding the product we have that the $k$-th coefficient of $\mathfrak{s} \cdot \mathfrak{a}$ is given by $\sum_{i=1}^N s_i \cdot \mathsf{NSgn}(k - i + 1) \cdot a_{(k-i \mod N)+1}$. Note that when working over the ring $\mathcal{R}_{N,q}$ defined by the cyclotomic polynomial $\Phi_N = X^N + 1$, we have that for $(k - i \mod N) + 1 \leq 0$, the sign of $a_{(k-i \mod N)+1}$ is reversed, thus we multiply the coefficient with $\mathsf{NSgn}(k - i + 1)$. Now if we set $\mathbf{a}'[i] = \mathsf{NSgn}(k - i + 1) \cdot a_{(k-i \mod N)+1}$, then the extracted sample is a valid LWE sample of $\mathsf{m}_k$ with respect to key $\mathbf{s}$. Finally, since the error in $e_k$ is a single coefficient of $e$, we have that $\mathcal{B}_{\mathsf{ct}} = \mathcal{B}$. □

*of lemma 7 - Correctness of Key Switching.* Let us first note that for all $i \in [n]$ we have

$$
\begin{aligned}
[b_i, \mathbf{a}_i^\top]^\top &= \sum_{j=1}^{\ell_{\mathsf{ksK}}} \mathbf{A}[i,j] \cdot \mathsf{ksK}[i,j] \\
&= \sum_{j=1}^{\ell_{\mathsf{ksK}}} \mathbf{A}[i,j] \cdot \mathsf{GLWE}_{\mathcal{B}_{\mathsf{ksK}},n',N,q}(\mathbf{s}', \mathbf{s}[i] \cdot \mathsf{L}_{\mathsf{ksK}}^j) \\
&= \mathsf{GLWE}_{\mathcal{B}_1,n',N,q}(\mathbf{s}', \mathbf{a}[i] \cdot \mathbf{s}[i]),
\end{aligned}
$$

where $\mathcal{B}_1 \leq \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}$. Then note that

$$
\begin{aligned}
[b', \mathbf{a}'^\top]^\top &= \sum_{i=1}^{n} \mathsf{GLWE}_{\mathcal{B}_1,n',N,q}(\mathbf{s}', \mathbf{a}[i] \cdot \mathbf{s}[i]) \\
&= \mathsf{GLWE}_{\mathcal{B}_2,n',N,q}(\mathbf{s}', \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathbf{s}[i]) \\
&= \mathsf{GLWE}_{\mathcal{B}_2,n',N,q}(\mathbf{s}', \mathbf{a}^\top \cdot \mathbf{s}),
\end{aligned}
$$

where $\mathcal{B}_2 \leq n \cdot \mathcal{B}_1 \leq n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}$. Let us denote $b = \mathbf{a}^\top \cdot \mathbf{s} + \mathsf{m} + e$ and $b' = \mathbf{a}'^\top \cdot \mathbf{s} + \mathbf{a}^\top \cdot \mathbf{s} + e'$, then

$$
\begin{aligned}
\mathbf{c}_{\mathsf{out}} &= [b, 0]^\top - [b', \mathbf{a}'^\top]^\top \\
&= [b - b', -\mathbf{a}'^\top]^\top \\
&= [-\mathbf{a}'^\top \cdot \mathbf{s}' + \mathsf{m} + e - e'].
\end{aligned}
$$

Hence, $\mathbf{c}_{\mathsf{out}}$ is a valid GLWE sample of $\mathsf{m}$ with respect to key $\mathbf{s}'$ and

$$
\begin{aligned}
||\mathsf{Var}(\mathsf{Error}(\mathbf{c}))||_\infty &\leq \mathcal{B} \\
&\leq ||e - e'||_2 \leq \mathcal{B}_{\mathbf{c}} + \mathcal{B}_2 \\
&\leq \mathcal{B}_{\mathbf{c}} + n \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{L}_{\mathsf{ksK}}^2 \cdot \mathcal{B}_{\mathsf{ksK}}.
\end{aligned}
$$

□

*of lemma 8 - Correctness of TFHE-Style Blind-Rotation.* Note that the blind rotation key $\mathsf{brK}$ consists of samples of bits: $\mathsf{RGSW}_{\mathcal{B}_{\mathsf{BR}},N,Q,\mathsf{L}_{\mathsf{RGSW}}}(\mathfrak{s}, \mathbf{Y}[i,j])$, where $\mathbf{Y}[i,j] \in \mathbb{B}$ for $i \in [n]$ and $j \in [u]$. Denote the current accumulator as $\mathsf{acc}_{curr} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{curr}},N,Q}(\mathfrak{s}, \mathsf{m}_{\mathsf{curr}})$ for some $\mathsf{m}_{\mathsf{curr}} \in \mathcal{R}_{N,Q}$. If we increment $\mathsf{acc}_{\mathsf{next}} \leftarrow \mathsf{Mux}\big(\mathsf{brK}[i,j], \mathsf{acc}_{\mathsf{curr}} \cdot X^{-\mathbf{a}[i] \cdot \mathbf{u}[j]}, \mathsf{acc}_{\mathsf{curr}}\big)$ for some $i \in [n]$ and $j \in [u]$, then from correctness of the Mux gate we have that

$\mathsf{acc}_{\mathsf{next}} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{next}},N,Q}(\mathfrak{s}, \mathsf{m}_{\mathsf{next}})$, where $\mathsf{m}_{\mathsf{next}} = \mathsf{m}_{\mathsf{curr}} \cdot X^{-\mathbf{a}[i]\cdot\mathbf{u}[j]\cdot K[i,j] \mod 2\cdot N} \in \mathcal{R}_{N,Q}$. Overall, we can represent the last message $\mathsf{m}_{\mathsf{out}}$ as

$$\begin{aligned}\mathsf{m}_{\mathsf{out}} &= \mathsf{m}_{\mathsf{acc}} \cdot \prod_{i=1}^{n}\prod_{j=1}^{u} X^{-\mathbf{a}[i]\cdot\mathbf{u}[j]\cdot K[i,j]} \\ &= \mathsf{m}_{\mathsf{acc}} \cdot \prod_{i=1}^{n} X^{-\mathbf{a}[i]\cdot\mathbf{s}[i]} \\ &= \mathsf{m}_{\mathsf{acc}} \cdot X^{-\mathbf{a}^\top\cdot\mathbf{s} \mod 2\cdot N}.\end{aligned}$$

From the analysis of the $\mathsf{Mux}$ gate we have that $\mathcal{B}_{\mathsf{next}} \leq \mathcal{B}_{\mathsf{curr}} + 2\cdot N\cdot \ell_{\mathsf{RGSW}}\cdot \mathsf{L}^2_{\mathsf{RGSW}}\cdot\mathcal{B}_{\mathsf{brK}}$. Finally since we perform $n\cdot u$ iterations, we have that

$$\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_{\mathsf{acc}} + 2\cdot n\cdot u\cdot N\cdot \ell_{\mathsf{RGSW}}\cdot \mathsf{L}^2_{\mathsf{RGSW}}\cdot\mathcal{B}_{\mathsf{brK}}.$$

□

*of theorem 1 - Correctness of TFHE Bootstrapping.* Let us denote $\mathsf{Phase}\big(\mathsf{ModSwitch}(\mathsf{ct}, q, 2N)\big) = \mathsf{m}' = \Delta_{2\cdot N,t}\cdot\mathsf{m} + e \mod 2N$. From correctness of modulus switching we have $||\mathsf{Var}(e)||_\infty \leq \mathcal{B}_N \leq \left(\frac{2\cdot N}{q}\right)^2\cdot\mathcal{B}_{\mathsf{ct}} + \mathsf{Ham}(\mathbf{s})\cdot||\mathsf{Var}(\mathbf{s})||_\infty$. From the correctness of blind rotation we have $\mathsf{acc}_{\mathsf{BR},F} \in \mathsf{RLWE}_{\mathcal{B}_{\mathsf{BR}},N,Q}(\mathfrak{s}, \mathsf{rotP}\cdot X^{\mathsf{m}' \mod 2\cdot N})$, where $\mathcal{B}_{\mathsf{BR}}$ is the bound induced by the blind rotation algorithm. From the assumption on $\mathsf{rotP}$, given that $\mathsf{m}' \leq N$ and the correctness of sample extraction we have $\mathbf{c}_Q \in \mathsf{LWE}_{\mathcal{B}_{\mathbf{c},Q},N,Q}(\mathbf{s}_F, \Delta_{Q,t}\cdot F(\mathsf{m}))$, where $\mathcal{B}_{\mathbf{c},Q} \leq \mathcal{B}_{\mathsf{BR}}$.

From correctness of key switching we have that $\mathbf{c}_{Q,\mathsf{ksK}} \in \mathsf{LWE}_{\mathcal{B},n,q}(\mathbf{s}, \Delta_{Q,t}\cdot F(\mathsf{m}))$, where $\mathcal{B}_{\mathbf{c},Q,\mathsf{ksK}} \leq \mathcal{B}_{\mathbf{c},Q} + \mathcal{B}_{KS}$, where $\mathcal{B}_{KS}$ is the bound induced by the key switching procedure. From correctness of modulus switching we have that $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\mathcal{B}_{\mathsf{out}},N,q}(\mathbf{s}, \Delta_{q,t}\cdot F(\mathsf{m}))$, where

$$\mathcal{B}_{\mathsf{out}} \leq \Big(\frac{q^2}{Q^2}\Big)\cdot\mathcal{B}_{\mathbf{c},Q,\mathsf{ksK}} + \mathsf{Ham}(\mathbf{s}_F)\cdot||\mathsf{Var}(\mathbf{s}_F)||_\infty.$$

To summarize we have:

$$\mathcal{B}_{\mathsf{out}} \leq \Big(\frac{q^2}{Q^2}\Big)\cdot(\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{KS}) + \mathsf{Ham}(\mathbf{s}_F)\cdot||\mathsf{Var}(\mathbf{s}_F)||_\infty,$$

where

$$\begin{aligned}\mathcal{B}_{\mathsf{BR}} &\leq 2\cdot n\cdot u\cdot N\cdot \ell_{\mathsf{RGSW}}\cdot \mathsf{L}^2_{\mathsf{RGSW}}\cdot\mathcal{B}_{\mathsf{brK}}, \text{ and} \\ \mathcal{B}_{KS} &\leq N\cdot\ell_{\mathsf{ksK}}\cdot\mathsf{L}^2_{\mathsf{ksK}}\cdot\mathcal{B}_{\mathsf{ksK}}.\end{aligned}$$

□