FDFB^2: Functional Bootstrapping via Sparse Polynomial Multiplication

Kamil Kluczniak[1] and Leonard Schild[2]

[1] secunet Security Networks kamil.kluczniak@gmail.com
[2] KU Leuven leonard.schild@esat.kuleuven.be

Abstract. Fully homomorphic encryption schemes are methods to perform computations over encrypted data. Since its introduction by Gentry, there has been a plethora of research optimizing the originally inefficient cryptosystems. Over time, different families have emerged. On the one hand, schemes such as BGV, BFV, or CKKS excel at performing coefficient-wise addition or multiplication over vectors of encrypted data. In contrast, accumulator-based schemes such as FHEW and TFHE provide efficient methods to evaluate boolean circuits and means to efficiently compute functions over small plaintext space of up to 4-5 bits in size.
In this paper, we focus on the second family. At a high level, accumulator-based schemes encode the range of a function f in the coefficients of a polynomial, which is then encrypted in a homomorphic accumulator. Given an input ciphertext, the coefficients of the encrypted polynomial are homomorphically rotated, such that there is a correspondence between the constant term of the result and the message contained in the ciphertext. In the end, it is possible to derive a ciphertext of the constant term encrypted with regard to the same encryption scheme as the input ciphertext. To summarize, by appropriately encoding the function f on the accumulated polynomial, we can compute f on the plaintext of the input ciphertext, where the output ciphertext has its noise magnitude independent of the input ciphertext. However, by default, it is necessary to impose restrictions on the type of functions we evaluate or drastically limit the plaintext space that can be correctly processed. Otherwise, the procedure's output will be incorrect and hard to predict.
In this work, we describe two novel algorithms that have no such restrictions. Furthermore, we derive an algorithm that enables a user to evaluate an arbitrary amount of functions at a computational cost that differs only by a constant amount compared to a single function. Our methods lead to an evaluation that is between 15 and 31% faster than previous works while also being conceptually simpler.

Keywords: Fully Homomorphic Encryption · Arbitrary Function Evaluation · Bootstrapping · FHE

1 Introduction

A fully homomorphic encryption scheme allows for performing any computation on encrypted data. The bootstrapping technique, initially introduced by Gentry [Gen09], is still the sole method for creating secure, fully homomorphic encryption schemes. This is because, in current homomorphic schemes, processing encrypted data generates noise, which can ultimately corrupt the plaintext if it becomes too excessive. In practical applications, bootstrapping also continues to be one of the primary efficiency bottlenecks.

By now, there are different types of fully homomorphic encryption schemes. On the one hand, schemes such as BGV [BV11, Bra12], BFV [FV12, BGV12] or CKKS [CKKS17] excel at performing coefficient-wise addition or multiplication over vectors of encrypted data. On the other hand, we have so-called circuit bootstrapping algorithms that excel at computing boolean circuits. To date, the best circuit bootstrapping algorithms are based on the blueprints by Alperin-Sheriff and Peikert [AP14], and the FHEW-style bootstrapping developed by Ducas and Micciancio [DM15]. The FHEW algorithm was further improved and applied in numerous works [CGGI16, CGGI20, CGGI17, MS18, BDF18, CIM19, GBA21, WWL$^+$24]. A core property that makes these algorithms efficient in applications is their ability to compute arbitrary functions on the input. Namely given an encryption ct of a message $m \in \mathbb{Z}_t$, one can bootstrap the ciphertext, and along the way compute a function $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ of the evaluators choice, given that f satisfies $f(m + t/2) = -f(m)$. The last restriction is often called the negacyclicity problem, since it stands in the way to freely chose the function f. Consequently, in order to compute an arbitrary function, a common technique is to assume that $m < t/2$. Some works like [JW22] call the technique "zero padding" the plaintext since we need to set the plaintext's most significant bit to zero. This way, we can exploit the bootstrapping algorithm to compute any lookup table. We call such bootstrapping algorithms functional or programmable bootstrapping. There are however some limitation to that technique:

1. The precision is small, typically between 3-4 bits, depending on the underlying parameters. We lose one bit of precision due to the padding being zero.

2. Due to limited plaintext space and zero padding, we cannot correctly compute many linear functions in the full domain of the plaintext space. That is, we cannot correctly compute linear functions in $\mathbb{Z}_t$. The reason is that adding or multiplying two ciphertexts with messages $< t/2$ may put the result over the $t/2$ threshold. As shown in [KS23, Klu22] this leads to speedups in the order of thousands when computing multivariate polynomials, in comparison to computing the same functions over a boolean circuit with the zero padding technique.

Two concurrent works [KS23, CLOT21] proposed extensions to the FHEW-style bootstrapping schemes to increase its precision and resolve the negacyclicity problem. Furthermore, [KS23] showed that one could exploit the function bootstrapping over the entire domain $\mathbb{Z}_t$ to compute multivariate polynomials with a larger plaintext space (even over 32 bits) using the residue number system over the plaintext space. This extends the repertoire of circuits efficiently computable by FHEW-style algorithms from only boolean to a mix of boolean and arithmetic functions without switching between BGV [BV11, Bra12] and BFV [FV12, BGV12] -style schemes. While to the best of our knowledge [CLOT21] was never implemented, nor parameters were proposed, [KS23] was implemented on top of the Palisade library, with timings over 10 seconds per bootstrapping depending on the precision. The main efficiency bottleneck in the algorithm is the necessity to compute multiple relatively expensive so-called blind rotation steps with less efficient parameters. In practice, the number of blind rotations is over 6. Micciancio and Polyakov [LMP22] and concurrently [YXS$^+$21] significantly improved the technique and effectively reduced the number of blind rotations to only two without significantly affecting other parameters. In practice, Micciancio and Polyakov's [LMP22] method takes approximately 0.9 seconds to compute a full domain bootstrap of a 5-bit plaintext. The method was further improved by [CBSZ23]. While [CBSZ23] requires four blind rotations, it allows to handle a plaintext space larger than [YXS$^+$21, LMP22]. To compare both methods for the same plaintext space, that is, when both methods have equivalent functionality, we must instantiate [YXS$^+$21, LMP22] with larger parameters. Nevertheless, [CBSZ23] reports only $1.2\times$ efficiency improvement. Finally, [CBSZ23] mentions a version of their algorithm that uses the multi-value technique from [CIM19], but to the best of our knowledge, there is no implementation of this method available.

1.1 Contribution

In this paper, we present improved algorithms for full domain functional bootstrapping that, similarly to Micciancio [LMP22], require only two blind rotation invocations but are able to handle a larger precision when instantiated over the same ring. Consequently, in comparison with the algorithm by Micciancio and Polyakov [LMP22], when targeting the same precision, our base algorithm achieves a speedup of about 31%. Our method also outperforms [CBSZ23] by approximately 15%.

We give two variants of our algorithm.

- The first variant computes a single function while bootstrapping a ciphertext. This variant is functionally equivalent to Micciancio and Polyakov [LMP22], but as stated earlier, we achieve a speedup of 31% while decreasing the evaluation key only by 7%. Compared to the faster version of [CBSZ23], we are 15% faster but with a key larger by 46%. Our overhead is due to an additional key switching key. Further, we discuss a version of the algorithm without that overhead. In absolute terms, our bootstrapping algorithm takes 0.59 seconds to bootstrap a 5-bit plaintext, while [LMP22] requires 0.87 seconds and [CBSZ23] requires 0.72 seconds.

- The second variant can compute an arbitrary number of functions on the same input ciphertext, at the expense of larger noise variance, which has to be compensated with slightly larger parameters. In particular, the evaluation key size increases by 29MB, which is an increase of 15%. The evaluation time remains roughly the same. Technically, we accomplish this by incorporating variants [MKG23] of the multi-value bootstrapping [CIM19]. This makes our algorithm particularly suitable for efficient digit decomposition and conversions between binary and arithmetic circuits. In contrast, prior works [KS23, LMP22, CBSZ23] had to execute one full domain functional bootstrapping per output digit, whereas we need to compute one full domain functional bootstrapping for all output. To illustrate the difference, let's compare ours with [CBSZ23] when computing binary decomposition of 5 bits. In our case, the computation takes roughly 0.7 seconds as we execute only one bootstrapping procedure, while for the faster variant of [CBSZ23], we do the same job in 3.5 seconds. That gives us a speedup of around 485%.

We implement, test, and integrate our algorithms into a publicly available open-source C++ library [fhe23]. Importantly, we integrate both our methods as well as [LMP22] into the library's high-level interface that allows programmers to use full-domain functional bootstrapping without needing to deal with the implementation details. Finally, we note that to the best of our knowledge we are the first to incorporate multi-value bootstrapping into a full-domain functional bootstrapping algorithm. We noticed that the algorithm by Micciancio and Polyakov [LMP22] can be extended this way as well. Hence, to give a better comparison between our core techniques, we also give parameters and implement a multi-value version of [LMP22]. Finally, we note that while [CBSZ23] uses a multi-value technique from [CIM19] as part of their faster construction, it doesn't seem possible to apply the technique to the output of the bootstrapping procedure.

Applications

The primary motivation in [KS23, CLOT21] is to build a full domain functional bootstrapping algorithm to extend the message space for input ciphertexts. Additionally, [KS23] used the technique to compute multivariate polynomials over a residue number system, effectively extending the plaintext space from 5 or 6 bits to 32 bits. Furthermore, [KS23] showed how to convert between arithmetic circuit and boolean circuit for FHEW-style schemes by exploiting the full domain bootstrapping algorithm to compute digit decomposition. Micciancio and Polyakov [LMP22] used their algorithm to compare large numbers. Further, Kluczniak [Klu22] instantiated Micciancio and Polyakov's [LMP22] method over NTRU ciphertexts to compute Gaussian elimination and solve systems of linear equations over encrypted data. Albrecht, Davidson, Deo, and Gardham [ADDG23] (to appear at Eurocrypt'24) use [LMP22] to construct oblivious pseudorandom functions. Finally, Deo et al. [DJL$^+$24] use full domain functional bootstrapping to evaluate learning with errors-based pseudorandom functions.

2 Preliminaries

In this section, we introduce the notation that will be used throughout this text and the background behind accumulator-based bootstrapping methods.

2.1 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. The symbol $\mathbb{Z}_q$ for $q \in \mathbb{N}$, refers to the quotient ring $\mathbb{Z}/q\mathbb{Z}$, that is, the integers modulo q. Furthermore we denote as $\mathcal{R}_Q$ the polynomial ring $\mathbb{Z}_Q[X]/(X^N+1)$ with $\log_2(N) \in \mathbb{N}$. When the value of Q is not ambiguous, we simply write $\mathcal{R}$. We denote vectors with a bold lowercase letter, e.g., $\mathbf{v}$, and matrices with uppercase letters $\mathbf{V}$. We refer to the ith entry of a vector $\mathbf{a}$ as $\mathbf{a}[i]$. Similarly, we address the (i,j)-th cell of a matrix $\mathbf{A}$ by $\mathbf{A}[i,j]$. For brevity, we denote as $[k]$ the sequence $\{0,1,...,k-1\}$. Finally, let $\mathfrak{a} = \sum_{i=0}^{N-1} \mathfrak{a}_i \cdot X^i$, be a polynomial with coefficients over any ring R, then we denote by $\mathsf{Coefs}(a)$ the coefficient vector $[\mathfrak{a}_i]_{i\in[N]} \in R^N$. 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 $\mathfrak{a} \in \mathcal{R}_Q$, we define $\mathsf{Var}(\mathfrak{a}) = \left[\mathsf{Var}(\mathsf{Coefs}(\mathfrak{a}))\right]_{i\in[N]}$, $\mathsf{stddev}(\mathfrak{a}) = \left[\mathsf{stddev}(\mathsf{Coefs}(\mathfrak{a}))\right]_{i\in[N]}$ and $\mathsf{E}(\mathfrak{a}) = \left[\mathsf{E}(\mathsf{Coefs}(\mathfrak{a}))\right]_{i\in[N]}$. In cases where $\forall i,j \in [N] : \mathsf{Var}(\mathfrak{a}_i) = \mathsf{Var}(\mathfrak{a}_j) = \sigma^2$ we write $\mathsf{Var}(\mathfrak{a}) = \sigma^2$ and do the same for standard deviation and expectation. By $\mathsf{Ham}(\mathbf{a})$ or $||\mathbf{a}||_0$ 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 $\Delta_{q,t} = \left\lfloor \frac{q}{t} \right\rceil$ and a rounding function for an element $\Delta_{q,t} \cdot a \in \mathbb{Z}_q$, as $\lfloor a \rceil_t^q = \left\lfloor \frac{t}{q} \cdot \Delta_{q,t} \cdot a \right\rceil$. For ring elements, we apply the rounding function coefficient-wise.

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 in the next subsection. For rings, we always use N to denote the degree of $\mathcal{R}_q$ or $\mathcal{R}_Q$. We denote bounds on variances of random variables by $\mathcal{B} \in \mathbb{R}$.

2.2 Learning With Errors

We recall the Regev encryption scheme [Reg10] that forms the base of the homomorphic encryption scheme we describe in the next sections. We are interested in two computational problems underlying the Regev cryptosystem. Namely the Learning with Errors (LWE) problem, and its structured ring variant. In order not to repeat ourselves, we recall a Generalized variant of the Regev cryptosystem and the underlying security assumption.

Definition 1 (Generalized Learning with Errors)**.** Let $\mathcal{D}_{\mathsf{sk}}$ be a (not necessarily uniform) distribution over $\mathcal{R}_Q$, and $\mathcal{D}_{\mathcal{R},\sigma}$ be a noise distribution over $\mathcal{R}_Q$ with standard deviation $\sigma > 0$, $n \in \mathbb{N}$ and $N \in \mathbb{N}$ be a power-of-two, that are chosen according to a security parameter λ. For $\mathfrak{a} \leftarrow_\$ \mathcal{R}_Q^n$, $\mathfrak{e} \leftarrow_\$ \mathcal{D}_{\mathcal{R},\sigma}$ and $\mathfrak{s} \in \mathcal{D}_{\mathsf{sk}}^n$, we define a GLWE sample of a message $\mathfrak{m} \in \mathcal{R}_Q$ with respect to $\mathfrak{s}$, as

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

Table 1: Key switching algorithms.

Key Switching	
$\mathsf{KeySwitchSetup}$	**Input:** Takes as input two LWE secret keys $\mathbf{s} \in \{0,1\}^n$, $\mathbf{s}' \in \mathbb{Z}_Q^N$, a performance parameter $L_{\mathsf{ksK}} \in \mathbb{N}$, and a standard deviation $\sigma_{\mathsf{ksK}} \in \mathbb{R}$. **Output:** Generates a key switching key ksK which consists of $N \cdot \log_{L_{\mathsf{ksK}}}(Q)$ LWE samples using $\mathbf{s}$ as a secret.
$\mathsf{KeySwitch}_{\mathbf{s}'\to\mathbf{s}}$	**Input:** Takes as input a key switching key ksK and a ciphertext $\mathsf{ct}' = \mathsf{LWE}(\mathbf{s}', m)$. **Output:** Returns a ciphertext $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, m)$. **Description:** The key switching process consists of $N \cdot \log_{L_{\mathsf{ksK}}}(Q)$ scalar multiplications in $\mathbb{Z}_Q$. The parameter L_{ksK} largely determines the time and space efficiency; that is, the larger L_{ksK}, the faster the computation and the smaller the space complexity of the key material, but the bigger the noise induced by the key switching operation.
$\mathsf{RLWEKeySwitchSetup}$	**Input:** Takes as input a LWE secret key $\mathbf{s}' \in \mathbb{Z}_Q^n$, a RLWE secret key $\mathfrak{s} \in \mathcal{R}_Q$, a performance parameter $L_{\mathsf{pK}} \in \mathbb{N}$, and a standard deviation $\sigma_{\mathsf{pK}} \in \mathbb{R}$. **Output:** Generates a key switching key pK which consists of $N/2 \cdot \log_{L_{\mathsf{pK}}}(Q)$ RLWE samples.
$\mathsf{RLWEKeySwitch}_{\mathbf{s}'\to\mathfrak{s}}$	**Input:** Takes as input a key switching key pK, a ciphertext $\mathsf{ct}' = \mathsf{LWE}(\mathbf{s}', m)$ and a polynomial $\mathfrak{h} \in \mathcal{R}_Q$. **Output:** Returns a ciphertext $\mathsf{ct} = \mathsf{RLWE}(\mathfrak{s}, m \cdot \mathfrak{h})$. **Description:** The purpose of the procedure is to switch from an LWE ciphertext, encoding the message m, to a RLWE ciphertext that encodes m in the constant coefficient. There are various techniques to implement such key switching procedure, in this paper we use the technique from [CDKS21]. The key switching process computes $\log(N)\log_{L_{\mathsf{pK}}}(Q)$ multiplications in $\mathcal{R}_Q$.

We say that the $\mathsf{GLWE}_{\sigma,n,N,Q}$-assumption holds if for any PPT adversary $\mathcal{A}$ we have

$$\left| \Pr\left[\mathcal{A}(\mathsf{GLWE}_{\sigma,n,N,Q}(\mathfrak{s},0))\right] - \Pr\left[\mathcal{A}(\mathcal{U}_Q^{n+1})\right] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{U}_Q^{n+1}$ is the uniform distribution over $\mathcal{R}_Q^{n+1}$.

We denote a LWE sample as $\mathsf{LWE}_{\sigma,n,Q}(\mathbf{s}, m) = \mathsf{GLWE}_{\sigma,n,1,Q}(\mathbf{s}, m)$, which is a special case of a GLWE sample where the ring is $\mathbb{Z}_Q[X]/(X+1)$. Note that, in this case, $\mathbf{s} \in \mathbb{Z}_p$ is a integer vector and $m \in \mathbb{Z}_p$. Similarly, we denote a RLWE sample as $\mathsf{RLWE}(\mathfrak{s}, \mathfrak{m}) = \mathsf{GLWE}_{\sigma,1,N,Q}(\mathfrak{s}, \mathfrak{m})$ which is the special case of an GLWE sample with $n = 1$. For simplicity, we omit to state the modulus and ring dimensions for and RLWE samples because we always use $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ where N is a power-of-two. Similarly, we usually omit the modulus and dimension for LWE samples, as those are Q and n respectively. We sometimes leave the inputs unspecified and substitute them with a dot $(\cdot)$ when it is not necessary to refer to them within the scope of a function. We define a spacial function $\mathsf{Phase}(\mathfrak{c}) = \langle \mathfrak{c}, [1, \mathbf{s}] \rangle$ on a ciphertext $\mathfrak{c} \in \mathcal{R}_Q^{(n+1)}$. The function Phase is always computed with respect to the ciphertexts secret key.

Table 2: List of sub-procedures commonly used in FHEW-like schemes [DM15] except for key switching procedures which are depicted by Table 1.

Blind Rotation	
$\mathsf{BlindRotateSetup}$	**Input:** Takes as input the LWE key $\mathsf{s} \in \{0,1\}^n$, a RLWE key $\mathfrak{s} \in \mathcal{R}_Q$, a performance parameter $L_{\mathsf{brK}} \in \mathbb{N}$, and a standard deviation σ_{brK}. **Output:** Generates a blind rotation key brK that consists of $2n \log_{L_{\mathsf{brK}}}(Q)$ RLWE samples under key $\mathfrak{s}$.
$\mathsf{BlindRotate}$	**Input:** Takes as input a blind rotation key brK, an LWE sample ct under modulus $2N$, and an accumulator $\mathsf{acc} = \mathsf{RLWE}(\mathfrak{s}, \mathfrak{p})$. **Output:** $\mathsf{BlindRotate}$ returns a sample $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}(\mathfrak{s}, \mathfrak{q})$ with $\mathfrak{q} = \mathfrak{p} \cdot X^{\mathsf{Phase}(\mathsf{ct})}$. In this paper, we consider the concrete blind rotation from [CGGI16], but other implementation are possible like [DM15, LMK+23]. The procedure from [CGGI16] takes $2n \cdot (\log_{L_{\mathsf{ksK}}}(Q) + 1)$ polynomial multiplications of elements in $\mathcal{R}_Q$.
Other	
$\mathsf{ModSwitch}_{Q \to q}$	**Input:** Takes as input an LWE sample $\mathsf{LWE}(\mathsf{s}, \Delta_{Q,t} \cdot \mathsf{m})$ under modulus Q. **Output:** Returns an LWE sample $\mathsf{LWE}(\mathsf{s}, \Delta_{q,t} \cdot \mathsf{m})$ under modulus q.
$\mathsf{SampleExt}$	**Input:** Takes as input a RLWE sample $\mathsf{RLWE}(\mathfrak{s}, \mathfrak{p})$ for $\mathfrak{p} \in \mathcal{R}_Q$ and an index $0 \le i < N$. **Output:** Returns an LWE sample $\mathsf{LWE}(\mathsf{s}', \mathsf{m})$ with $\mathsf{m} = \mathfrak{m}[i]$ that is, the LWE sample encodes the i-th coefficient of the polynomial $\mathfrak{m}$. s' is a vector of dimension N such that $\mathsf{s}'_i = \mathfrak{s}_i$

2.3 Computing on Encrypted Data

For completeness we recall the definition of Fully Homomorphic Encryption. Furthermore, as our main contribution presented in Section 3 relies on a set of procedures introduced in previous work, we give all high-level interfaces at Table 1 and Table 2. In this paper we use these procedures in a black-box fashion stating only their input-output relation. In particular, we note that there may be different concrete implementations of these primitives offering various time-memory-correctness trade-offs. Nevertheless, we can state our main contribution and its analysis without referring to a concrete realization of these primitives, what we believe makes the presentation accessible to non-experts. But for our implementation, noise estimation and experiments in Section 4 we give an efficient sample instantiation of these procedures.

2.4 Fully Homomorphic Encryption

An FHE scheme consists of four algorithms (Setup, Enc, Eval, Dec), each with the following syntax [RAD+78, Gen09].

- $\mathsf{Setup}(\lambda)$: This algorithm takes as input a security parameter λ and outputs an evaluation key ek and a secret key sk.
- $\mathsf{Enc}(\mathsf{sk}, \mathsf{m})$: This algorithm takes as input a secret key sk as well as a message m and returns a ciphertext ct.

- $\mathsf{Eval}(\mathsf{ek}, [\mathsf{ct}_i]_{i=1}^n, \mathcal{C})$: Given an evaluation key ek, ciphertexts $[\mathsf{ct}_i]_{i=1}^n$, and a circuit $\mathcal{C}$, this (non-)deterministic algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Given a secret key sk and a ciphertext ct, this deterministic algorithm outputs a message m.

Informally, we say that an FHE scheme is correct, if the outcome of the evaluation of a circuit $\mathcal{C}$ on ciphertexts encrypting messages $\mathsf{m}_1, \dots, \mathsf{m}_n$ decrypts to $\mathcal{C}(\mathsf{m}_1, \dots, \mathsf{m}_n)$. Formally, we say that FHE is correct if for all security parameters $\lambda \in \mathbb{N}$, the circuits $\mathcal{C} : \mathcal{M}^n \to \mathcal{M}$ over the message space $\mathcal{M}$ of depth $\mathsf{poly}(\lambda)$, and all messages $[\mathsf{m}_i \in \mathcal{M}]_{i=1}^n$ we have

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) = \mathcal{C}([\mathsf{m}_i]_{i=1}^n)\right] = 1 - \mathsf{negl}(\lambda),$$

where $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\lambda)$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_i) = \mathsf{m}_i$ for all $i \in [n]$ and $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Eval}(\mathsf{ek}, [\mathsf{ct}_i]_{i=1}^n, \mathcal{C})$. For efficiency, we require that Setup, Enc and Dec run in polynomial time in the security parameter, that is $\mathsf{poly}(\lambda)$, and Eval runs in $\mathsf{poly}(\lambda, |\mathcal{C}|)$. Finally, we say that an FHE scheme is compact if the size of the output of Eval is independent of the size of the circuit $\mathcal{C}$. More specifically, we require that $|\mathsf{Eval}(\mathsf{ek}, [\mathsf{ct}_i]_{i=1}^n, \mathcal{C})|$ is $\mathsf{poly}(\lambda, |\mathcal{M}|)$.

Indistinguishability Under Chosen Plaintext Attack. Let $\lambda \in \mathbb{N}$ be a security parameter and $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a probabilistic polynomial-time algorithm (adversary). We say that an FHE scheme is **IND-CPA**-secure if the probability

$$\Pr\left[\mathcal{A}_1(\mathsf{ct}_b, \mathsf{st}) = b \colon \begin{array}{c} \mathsf{sk} \leftarrow \mathsf{Setup}(\lambda), \\ (\mathsf{st}, \mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}_0^{\mathcal{O}(\mathsf{sk},.)}(\lambda), \\ b \leftarrow_{\$} \{0,1\}, \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\lambda, \mathsf{sk}, \mathsf{m}_b) \end{array}\right]$$

is at most $\mathsf{negl}(\lambda)$ for all probabilistic polynomial-time adversaries $\mathcal{A}$; the oracle $\mathcal{O}$ on input of a message m outputs $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathsf{m})$.

3 Our Functional Bootstrapping

In this section, we introduce our novel bootstrapping algorithm that is not restricted to the evaluation of negacyclic functions. We present our algorithm for the setting where a single function is to be evaluated in Subsection 3.1. Then, in Subsection 3.2, we generalize our algorithm to allow the computation of multiple functions at the cost of only a single bootstrapping. That is, we give a multi-value bootstrapping variant of our full-domain functional bootstrapping algorithm.

3.1 Single Function Case

First, we recall a helper function at Algorithm 1 that compute so-called rotation or test polynomials rotP_0, $\mathsf{rotP}_1 \in \mathcal{R}$. These polynomials are then used in Algorithm 3 as well as in Algorithm 4 to homomorphically rotate the encrypted accumulator and compute a function F. In other words, polynomials are chosen and applied such that the resulting RLWE ciphertext contains $F(m)$ at its constant coefficient, where m is the plaintext in the ciphertext input to the bootstrapping procedure. Finally, we can extract an LWE ciphertext encoding the constant coefficient, i.e., $F(m)$, and use the ciphertext to compute a linear function or another bootstrapping.

At Algorithm 2, we give the Setup algorithm that generates the evaluation key and the secret key for the FHE scheme. In short, the algorithm generates LWE-to-LWE, RLWE-to-LWE key switching keys, and a blind rotation key, which are essential to realize our bootstrapping algorithms.

Algorithm 1: $\mathsf{Setup}(F)$

```
    Input: A map $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$
    Output: Polynomials $\mathsf{rotP}_0, \mathsf{rotP}_1$
1 $\mathsf{rotP}_0 \leftarrow \mathbf{0} \in \mathcal{R}$;
2 $\mathsf{rotP}_1 \leftarrow \mathbf{0} \in \mathcal{R}$;
3 for $i = 0$ to $\frac{t}{2} - 1$ do
4     $\mathsf{rotP}_0[\lfloor 2\frac{N}{t} \rceil \cdot i] \leftarrow -F\big(\frac{t}{2} - i - 1\big) \pmod{t}$;
5     $\mathsf{rotP}_1[\lfloor 2\frac{N}{t} \rceil \cdot i] \leftarrow F(t - i - 1)$;
6 end
7 return $\mathsf{rotP}_0, \mathsf{rotP}_1$
```

Algorithm 2: $\mathsf{Setup}(\lambda)$

```
    Input: A security parameter $\lambda$.
    Input: A function $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ to be evaluated during bootstrapping
    Output: A evaluation key $\mathsf{ek}$ and a secret key $\mathsf{sk}$.
1 Choose $\mathbf{s} \in \{0,1\}^n$ with $n$ depending on $\lambda$. ;
    /* We use secret keys with binary entries due to the blind rotation from [CGGI16],
       but it can be ternary, or Gaussian as in [DM15, LMK+23]. */
2 Choose $\mathfrak{s} \in \mathcal{D}_{\mathsf{sk}}$, where $\mathcal{D}_{\mathsf{sk}}$ is some secret key distribution dependent on $\lambda$.;
3 $\mathsf{s}' \leftarrow \mathsf{Coefs}(\mathfrak{s})$.;
4 $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\mathsf{s}', \mathbf{s}, L_{\mathsf{ksK}}, \sigma_{\mathsf{ksK}})$, where $L_{\mathsf{ksK}} \in \mathbb{N}$ is a performance
    parameter and $\sigma_{\mathsf{ksK}} \in \mathbb{R}$ is a standard deviation dependent on $\lambda$.;
5 $\mathsf{pK} \leftarrow \mathsf{RLWEKeySwitchSetup}(\mathbf{s}, \mathfrak{s}, L_{\mathsf{pK}}, \sigma_{\mathsf{pK}})$, where $L_{\mathsf{pK}} \in \mathbb{N}$ is a performance
    parameter and $\sigma_{\mathsf{pK}} \in \mathbb{R}$ is a standard deviation dependent on $\lambda$.;
6 $\mathsf{brK} \leftarrow \mathsf{BlindRotateSetup}(\mathbf{s}, \mathfrak{s}, L_{\mathsf{brK}}, \sigma_{\mathsf{brK}})$, where $L_{\mathsf{brK}} \in \mathbb{N}$ is a performance
    parameter and $\sigma_{\mathsf{brK}} \in \mathbb{R}$ is a standard deviation dependent on $\lambda$.;
7 $\mathsf{ek} \leftarrow \big[\mathsf{ksK}, \mathsf{pK}, \mathsf{brK}\big]$;
8 $\mathsf{sk} \leftarrow [\mathsf{s}']$;
9 return $\mathsf{ek}, \mathsf{sk}$
```

At Algorithm 3, we give our novel bootstrapping algorithm. At a high level, our algorithm works as follows. Initially, we set the accumulator to a polynomial such that, after performing the first blind rotation, we obtain an RLWE sample encrypting a polynomial $\mathfrak{r}$ with a constant term equal $\mathfrak{r}_0 = -\left\lfloor \frac{Q}{2t} \right\rceil$ if $m < \frac{t}{2}$ and $\mathfrak{r}_0 = \left\lfloor \frac{Q}{2t} \right\rceil$ otherwise. After applying the blind rotation, we extract a LWE sample containing the constant coefficient of the accumulator and add a scalar $\left\lfloor \frac{Q}{2t} \right\rceil$ in order to map $(-1)^{1-b} \left\lfloor \frac{Q}{2t} \right\rceil$ to the $b \cdot \left\lfloor \frac{Q}{t} \right\rceil$ with $b = 0$ when $m < \frac{t}{2}$ and $b = 1$ otherwise. Then, we key-switch the LWE sample to an RLWE sample which encrypts a polynomial

$$\mathfrak{t} = b \cdot \left\lfloor \frac{Q}{t} \right\rceil \cdot \mathfrak{h} = b \cdot \left\lfloor \frac{Q}{t} \right\rceil \sum_{i=0}^{\frac{2N}{t}-1} X^i.$$

As the value of b depends on the most significant bit message, we construct a new accumulator with the correct sign flip by multiplying the RLWE sample with the polynomials created by Algorithm 2. Finally, we blind-rotate the derived accumulator, and the correct output will be encoded in the constant term of the result, from which we derive a LWE sample.

```
Algorithm 3: Bootstrap($\mathsf{ct}, F$)
    Input: The evaluation key $\mathsf{ek} = (\mathsf{brK}, \mathsf{ksK}, \mathsf{pK})$
    Input: A ciphertext $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t} m)$
    Input: A function $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ to be evaluated during bootstrapping
    Output: $\mathsf{ct}_{\mathsf{out}}$
    /* Recall pre-computed values */
 1  $\mathfrak{h} = \sum_{i=0}^{\frac{2N}{t}-1} X^i$;
 2  $\mathsf{s}' = \mathsf{Coefs}(\mathfrak{s})$;
 3  $\mathsf{ct}_{Q_{\mathsf{ksK}}} \leftarrow \mathsf{ModSwitch}_{Q \to Q_{\mathsf{ksK}}}(\mathsf{ct})$;
 4  $\mathsf{ct}_{\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}_{\mathsf{s}' \to \mathbf{s}}(\mathsf{ksK}, \mathsf{ct}_{Q_{\mathsf{ksK}}})$;
 5  $\mathsf{ct}_{2N} \leftarrow \mathsf{ModSwitch}_{Q \to 2N}(\mathsf{ct}_{\mathsf{ksK}})$;
 6  $\mathsf{ct}_{\mathsf{boot}} \leftarrow \mathsf{ct}_{2N} + \frac{N}{t}$;
 7  $\mathsf{sgnP} \leftarrow \left\lfloor \frac{Q}{2t} \right\rceil \cdot \left(-1 + \sum_{i=1}^{N-1} X^i\right) \in \mathcal{R}_Q$;
 8  $\mathsf{acc}_{\mathsf{sgn,in}} \leftarrow [\mathsf{sgnP}, \mathbf{0}]^\top$;
 9  $\mathsf{acc}_{\mathsf{sgn,out}} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_{\mathsf{sgn,in}}, \mathsf{ct}_{\mathsf{boot}})$;
    /* Extract and map $\pm \left\lfloor \frac{Q}{2t} \right\rceil$ to $\left\{0, \left\lfloor \frac{Q}{t} \right\rceil\right\}$ */
10  $\mathsf{ct}_{\mathsf{sgn}} \leftarrow \mathsf{SampleExt}(\mathsf{acc}_{\mathsf{sgn,out}}, 0) + \left\lfloor \frac{Q}{2t} \right\rceil$;
    /* RLWE Key-switch with padding */
11  $\mathsf{acc} \leftarrow \mathsf{RLWEKeySwitch}_{\mathsf{s}' \to \mathfrak{s}}(\mathsf{pK}, \mathsf{ct}_{\mathsf{sgn}}, \mathfrak{h})$;
12  $\mathsf{rotP}_0, \mathsf{rotP}_1 \leftarrow \mathsf{Setup}(F)$;
13  $\mathsf{acc}_{F,\mathsf{in}} \leftarrow \mathsf{acc} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0) + \mathfrak{h} \cdot \left\lfloor \frac{Q}{t} \right\rceil \cdot \mathsf{rotP}_0$;
14  $\mathsf{acc}_{F,\mathsf{out}} \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_{F,\mathsf{in}}, \mathsf{ct}_{\mathsf{boot}})$;
15  $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{SampleExt}(\mathsf{acc}_{F,\mathsf{out}}, 0)$;
16  return $\mathsf{ct}_{\mathsf{out}}$
```

By including the polynomial $\mathfrak{h}$ in the RLWE key-switching key, we incorporate the padding necessary to deal with the LWE error at an early point. Then, the polynomials $\mathsf{rotP}_0, \mathsf{rotP}_1$ do not need to be dense, and only $\frac{t}{2}$ coefficients need to be set. Consequently, multiplying an RLWE sample with such polynomials increases the noise variance by a factor of $\frac{t^3}{2}$ instead of Nt^2 which is beneficial as $N > t$ and often even $N > t^2$.

We state the bound on the output variance in Theorem 1

Theorem 1 (Correctness of functional bootstrapping)**.** *Let* ct *be a LWE sample under modulus Q and dimension N such that* $\mathsf{Phase}(\mathsf{ct}) = \Delta_{Q,t} m + e_Q$*. Furthermore, let* ct_{2N} *be the LWE sample obtained after key- and modulus switching* ct *to a dimension n and modulus equal to $2N$. Let $\mathcal{B}_{\mathsf{BR}}$ be the variance of noise polynomial introduced by the blind-rotation and let $\mathcal{B}_{\mathsf{pK}}$ be a bound on the variance of the noise created by LWE to RLWE key-switching.*

If $\left|\mathsf{Phase}(\mathsf{ct}_{2N} - \Delta_{2N,t} m)\right| = e \in \left(-\frac{N}{t}, \frac{N}{t}\right) \bigcap \mathbb{Z}$*, then Algorithm 3, on input a ciphertext* $\mathsf{ct} \in \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t} m)$ *and a function $F : \mathbb{Z}_t \mapsto \mathbb{Z}_t$, outputs a LWE sample* $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t} \cdot F(m))$ *with noise variance $\mathcal{B}_{\mathsf{out}}$ and*

$$\mathcal{B}_{\mathsf{out}} \leq (\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{\mathsf{pK}}) \cdot \left(\frac{t^3}{2} + 1\right).$$

Proof. Recall that $\mathsf{Phase}(\mathsf{ct}_{2N}) = \Delta_{2N,t} \cdot m + e = \left\lfloor \frac{2N}{t} \right\rceil m + e \pmod{q}$, i.e. messages are distributed around multiples of $\frac{2N}{t}$. Furthermore, we have that $e \in \left(-\frac{N}{t}, \frac{N}{t}\right) \bigcap \mathbb{Z}$ and we write

$$\begin{aligned}\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) &= \mathsf{Phase}(\mathsf{ct}_{2N}) + \frac{N}{t}\\ &= \Delta_{2N,t}m + e + \frac{N}{t}\\ &= \frac{2N}{t}\left(\frac{t}{2}b + w\right) + \bar{e}\\ &= Nb + \frac{2N}{t}w + \bar{e} \pmod{2N}\end{aligned}$$

with $b \in \{0,1\}$, $0 \leq w < \frac{t}{2}$ and $1 \leq \bar{e} < \frac{2N}{t}$. The latter follows from the initial assumption that w.r.t e. Then, we can note that

$$m \geq \frac{t}{2} \Leftrightarrow \mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) \geq N \Leftrightarrow b = 1$$

By the properties of blind-rotation,it follows that

$$\mathsf{Phase}(\mathsf{acc}_{\mathsf{sgn},\mathsf{out}}) = \left\lfloor \frac{Q}{2t} \right\rceil \cdot \left(1 - \sum_{i=1}^{N-1} X^i\right) \cdot X^{\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}})} + \mathfrak{e}_{\mathsf{out}}^{(0)}$$

where $\mathfrak{e}_{\mathsf{out}}^{(0)}$ is the error polynomial induced by blindrotation, and more importantly

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}_{\mathsf{sgn},\mathsf{out}})_0 &= \left\lfloor \frac{Q}{2t} \right\rceil \cdot \left(\left(1 - \sum_{i=1}^{N-1} X^i\right) \cdot X^{\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}})}\right)_0 + \mathfrak{e}_{\mathsf{out},0}\\ &= \begin{cases} -\left\lfloor \frac{Q}{2t} \right\rceil + \mathfrak{e}_{\mathsf{out},0} & \text{if } \mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) < N\\ \left\lfloor \frac{Q}{2t} \right\rceil + \mathfrak{e}_{\mathsf{out},0} & \text{if } \mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) \geq N \end{cases}\\ &= (-1)^{1-b}\left\lfloor \frac{Q}{2t} \right\rceil + \mathfrak{e}_{\mathsf{out},0}\end{aligned}$$

Then, after extracting the constant coefficient, adding $\left\lfloor \frac{Q}{2t} \right\rceil$ and key-switching to RLWE, we obtain a RLWE sample acc with

$$\mathsf{Phase}(\mathsf{acc}) = \mathfrak{h}\left(\left\lfloor \frac{Q}{t} \right\rceil b + \mathfrak{e}_{\mathsf{out},0}\right) + \mathfrak{e}_{\mathsf{RLWE}}$$

where $\mathfrak{e}_{\mathsf{RLWE}}$ is the error polynomial stemming from RLWE keyswitching. Multiplying with $\mathsf{rotP}_0, \mathsf{rotP}_1$, we have that

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}_{F,\mathsf{in}}) &= \begin{cases} \left\lfloor \frac{Q}{t} \right\rceil \mathfrak{h} \cdot \mathsf{rotP}_0 + \bar{\mathfrak{e}} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0) & \text{if } b = 0\\ \left\lfloor \frac{Q}{t} \right\rceil \mathfrak{h} \cdot \mathsf{rotP}_1 + \bar{\mathfrak{e}} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0) & \text{if } b = 1 \end{cases}\\ &= \left\lfloor \frac{Q}{t} \right\rceil \mathfrak{h} \cdot \mathsf{rotP}_b + \bar{\mathfrak{e}} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0)\end{aligned}$$

with $\bar{\mathfrak{e}} = (\mathfrak{h} \cdot \mathfrak{e}_{\mathsf{out},0} + \mathfrak{e}_{\mathsf{RLWE}})$. Note that $\mathfrak{e}_{\mathsf{out},0}$ is a constant polynomial or scalar, so multiplying by $\mathfrak{h}$ will not increase the bound on the overall coefficient variance. Finally, it follows that

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}_{F,\mathsf{out}}) &= \mathsf{Phase}(\mathsf{acc}_{F,\mathsf{in}}) \cdot X^{\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}})} + \mathfrak{e}_{\mathsf{out}}^{(1)}\\ &= (-1)^{(1-b)}\left\lfloor \frac{Q}{t} \right\rceil \mathfrak{h} \cdot \mathsf{rotP}_b \cdot X^{\frac{2N}{t}w + \bar{e}} + \bar{\mathfrak{e}} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0) + \mathfrak{e}_{\mathsf{out}}^{(1)}\end{aligned}$$

where $\mathfrak{e}_{\mathsf{out}}^{(1)}$ is the error polynomial induced by the second blind-rotation. Note that the product $\mathfrak{h} \cdot \mathsf{rotP}_b$ yields a polynomial encoding the values of the function F in chunks of size $\frac{2N}{t}$ and in reversed order from $F(t-1)$ to $F(0)$. As we assume that $1 \leq \bar{e} < \frac{2N}{t}$, this error is smaller than the chunk size, and since the selection of the polynomials $\mathsf{rotP}_b, b \in \{0,1\}$ fully depends on the most significant bit of the phase we are able to encode the function values accordingly. Then, correctness follows and we have that

$$\mathsf{Phase}(\mathsf{acc}_{F,\mathsf{out}})_0 = \left\lfloor \frac{Q}{t} \right\rceil \cdot F(m) + (\bar{\mathfrak{e}} \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0) + \mathfrak{e}_{\mathsf{BR}}^{(1)})_0$$

Tracing our steps backwards, we note

$$\begin{aligned}\mathsf{Var}(\mathsf{Phase}(\mathsf{acc}_{F,\mathsf{out}})) &\leq \mathsf{Var}(\bar{\mathfrak{e}}(\mathsf{rotP}_1 - \mathsf{rotP}_0)) + \mathcal{B}_{\mathsf{BR}} \\ &\leq \mathsf{Var}(\bar{\mathfrak{e}}) \cdot t^2 \cdot \frac{t}{2} + \mathcal{B}_{\mathsf{BR}} \\ &\leq (\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{\mathsf{pK}}) \cdot \frac{t^3}{2} + \mathcal{B}_{\mathsf{BR}} \\ &\leq (\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{\mathsf{pK}}) \cdot \left(\frac{t^3}{2} + 1\right)\end{aligned}$$

□

3.2 Amortization over Multiple Functions

We have previously discussed how to evaluate an arbitrary function homomorphically while circumventing the issue of negacyclicity. However, we can expect that in some cases, we need to evaluate several functions using the same ciphertext as input. One such case is given in a setting in which we aim to decompose a message into several bits. A naive solution to this problem would consist of applying Algorithm 3 for every function, which is not an efficient way to proceed. In this section, we describe a modification of the previous algorithm that allows the evaluation of an arbitrary amount of function at the cost of a slightly higher output variance.

We begin by briefly discussing challenges that arise in the multiple-function setting as opposed to the single-function case. In Algorithm 3, the initial blind rotation serves the purpose of determining the most significant bit of the phase, such that the accumulator of the second blind rotation can be set up to properly take the sign flip of the procedure into account and map values correctly. If we aim to bootstrap several functions, one option would consist of determining the sign as previously and repeating the next stages for every function. While correct, the number of blind rotations is linear in the number of functions, which is not optimal in terms of performance. We can note that the first stage of such an approach only determined information about the domain in which the phase of the LWE sample falls into, i.e., whether the phase is contained in $[0, N)$ or $[N, 2N)$. However, in order to compute several functions in a constant number of blind rotations, it will also be necessary to extract information about the phase modulo N so that we can multiply the function polynomials by the appropriate monomial.

At a high level, we solve this issue by determining RLWE samples $\mathsf{acc}_+ = \mathsf{RLWE}(\mathfrak{s}, \mathfrak{p})$, $\mathsf{acc}_- = \mathsf{RLWE}(\mathfrak{s}, \mathfrak{q})$, such that

$$\mathfrak{p} = X^{\mathsf{Phase}(\mathsf{ct})} \pmod{N}$$

and

$$\mathfrak{q} = (-1)^{(1-b)} X^{\mathsf{Phase}(\mathsf{ct})} \pmod{N}$$

where $b = 0$ if the encrypted message m is less than $\frac{t}{2}$ and $b = 1$ otherwise. Then, if $\mathsf{rotP}_0, \mathsf{rotP}_1$ are the polynomials obtained by Algorithm 2, we can compute

$$\mathsf{acc} = \mathsf{acc}_+ \cdot (\mathsf{rotP}_1 + \mathsf{rotP}_0) + \mathsf{acc}_- \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0)$$

obtaining $\mathsf{Phase}(\mathsf{acc}) = 2 \cdot \mathsf{rotP}_b \cdot X^{\mathsf{Phase}(\mathsf{ct}) \pmod N}$. Once again, the choice of polynomial rotP_b is determined by the most significant bit of the message, and the multiplication by 2 can be counteracted by employing an appropriate scaling factor. Furthermore, once $\mathsf{acc}_-, \mathsf{acc}_+$ have been determined, we can repeat this calculation for arbitrarily many functions or polynomials.

We give our solution in Algorithm 4 and state the correctness and the bound on the output noise in Theorem 2.

Algorithm 4: Bootstrap-MV

```
Input: The evaluation key $\mathsf{ek} = (\mathsf{brK}, \mathsf{ksK}, \mathsf{pK})$
Input: A ciphertext $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t}m)$
Input: Functions $(F_j)_{j\in[k]}$
Output: $(\mathsf{ct}^{(j)}_{\mathsf{out}})_{j\in[k]}$ such that $\mathsf{ct}^{(j)}_{\mathsf{out}} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t}F_j(m))$
/* Recall pre-computed values */
1  $\mathfrak{h} = \sum_{i=0}^{\frac{2N}{t}-1} X^i$;
2  $\mathsf{s}' = \mathsf{Coefs}(\mathfrak{s})$;
3  $\mathsf{ct}_{Q_{\mathsf{ksK}}} \leftarrow \mathsf{ModSwitch}_{Q\to Q_{\mathsf{ksK}}}(\mathsf{ct})$;
4  $\mathsf{ct}_{\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}_{\mathsf{s}'\to\mathbf{s}}(\mathsf{ksK}, \mathsf{ct}_{Q_{\mathsf{ksK}}})$;
5  $\mathsf{ct}_{2N} \leftarrow \mathsf{ModSwitch}_{Q\to 2N}(\mathsf{ct}_{\mathsf{ksK}})$;
6  $\mathsf{ct}_{\mathsf{boot}} \leftarrow \mathsf{ct}_{2N} + \frac{N}{t}$;
7  $\mathsf{sgnP} \leftarrow \left\lfloor \frac{Q}{2t} \right\rceil \cdot \mathfrak{h}$;
8  $\mathsf{acc}_{\mathsf{sgn}} \leftarrow [\mathsf{sgnP}, \mathbf{0}]^\top$;
9  $\mathsf{acc}_- \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_{\mathsf{sgn}}, \mathsf{ct}_{\mathsf{boot}})$ ;
10 for $i = 0$ to $\frac{t}{2} - 1$ do
11     $\mathsf{ct}^{(i)} \leftarrow \mathsf{SampleExt}(\mathsf{acc}_-, \frac{2N}{t} i)$;
12 end
13 $\mathsf{acc}_{\mathsf{BR}} \leftarrow \mathsf{RLWEKeySwitch}_{\mathsf{s}'\to\mathfrak{s}}\left(\mathsf{pK}, \sum_{i=0}^{\frac{t}{2}-1} \mathsf{ct}^{(i)}, \mathfrak{h}\right)$;
14 $\mathsf{acc}_+ \leftarrow \mathsf{BlindRotate}(\mathsf{brK}, \mathsf{acc}_{\mathsf{BR}}, \mathsf{ct}_{\mathsf{boot}})$;
15 for $j = 0$ to $k - 1$ do
16     $\mathsf{rotP}_0, \mathsf{rotP}_1 \leftarrow \mathsf{Setup}(F_j)$;
17     $\mathsf{acc}^{(j)} \leftarrow \mathsf{acc}_+ \cdot (\mathsf{rotP}_1 + \mathsf{rotP}_0) + \mathsf{acc}_- \cdot (\mathsf{rotP}_1 - \mathsf{rotP}_0)$;
18     $\mathsf{ct}^{(j)}_{\mathsf{out}} \leftarrow \mathsf{SampleExt}(\mathsf{acc}^{(j)}, 0)$;
19 end
20 return $\left[\mathsf{ct}^{(j)}_{\mathsf{out}}\right]_{j\in[k]}$
```

Theorem 2 (Output Noise Variance of Algorithm 4)**.** *Let* ct *be a LWE sample under modulus Q and dimension N such that* $\mathsf{Phase}(\mathsf{ct}) = \Delta_{Q,t}m + e_Q$. *Furthermore, let* ct_{2N} *be the LWE sample obtained after key- and modulus switching* ct *to a dimension n and modulus equal to $2N$. Let $\mathcal{B}_{\mathsf{BR}}$ be the variance of noise polynomial introduced by the blind-rotation and let $\mathcal{B}_{\mathsf{pK}}$ be a bound on the variance of the noise created by LWE to RLWE key-switching.*

If $|\mathsf{Phase}(\mathsf{ct}_{2N} - \Delta_{2N,t}m)| = e \in \left(-\frac{N}{t}, \frac{N}{t}\right) \bigcap \mathbb{Z}$, *then Algorithm 3, on input a ciphertext* $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t}m)$ *and a set of functions $F_j : \mathbb{Z}_t \mapsto \mathbb{Z}_t, j \in [k]$, outputs LWE samples* $\mathsf{ct}^{(j)}_{\mathsf{out}} = \mathsf{LWE}(\mathbf{s}, \Delta_{Q,t} \cdot F_j(m))$ *with noise variance $\mathcal{B}_{\mathsf{out}}$ and*

$$\mathcal{B}_{\mathsf{out}} \leq \left(t^3 + \frac{t^4}{4}\right)\mathcal{B}_{\mathsf{BR}} + \frac{t^3}{2}\mathcal{B}_{\mathsf{pK}}.$$

Proof. As in the proof of Algorithm 3, we note $\mathsf{Phase}(\mathsf{ct}_{2N}) = \Delta_{2N,t} \cdot m + e = \left\lfloor \frac{2N}{t} \right\rceil m + e \pmod q$, i.e. messages are distributed around multiples of $\frac{2N}{t}$. Furthermore, we have that $e \in \left(-\frac{N}{t}, \frac{N}{t}\right) \bigcap \mathbb{Z}$ and we write

$$\begin{aligned}\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) &= \mathsf{Phase}(\mathsf{ct}_{2N}) + \frac{N}{t}\\ &= \Delta_{2N,t} m + e + \frac{N}{t}\\ &= \frac{2N}{t}\left(\frac{t}{2}b + w\right) + \bar{e}\\ &= Nb + \frac{2N}{t}w + \bar{e} \pmod{2N}\end{aligned}$$

with $b \in \{0,1\}$, $0 \le w < \frac{t}{2}$ and $1 \le \bar{e} < \frac{2N}{t}$. The latter follows from the initial assumption that w.r.t e. Then, it holds again that that

$$m \ge \frac{t}{2} \Leftrightarrow \mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}}) \ge N \Leftrightarrow b = 1$$

Next, observe that for acc_- in Algorithm 4

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}_-) &= \left\lfloor \frac{Q}{2t} \right\rceil \cdot \mathfrak{h} \cdot X^{\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}})} + \mathfrak{e}_+\\ &= (-1)^{1-b} \left\lfloor \frac{Q}{2t} \right\rceil \cdot \mathfrak{h} \cdot X^{\frac{2N}{t}w + \bar{e}} + \mathfrak{e}_+\end{aligned}$$

where $\mathfrak{e}_+$ is the error term from blind-rotation. Disregarding $\mathfrak{e}_+$ at first, it follows that the coefficients of the polynomial contained in acc_- whose index is a multiple of $\frac{2N}{t}$ are all zero except a single one containing $\pm \left\lfloor \frac{Q}{2t} \right\rceil$ and therefore adding all slots via sample extraction yields a LWE sample of $(-1)^{1-b} \left\lfloor \frac{Q}{2t} \right\rceil$, i.e. a value induced by b. In the next step, we key-switch this sample to a RLWE with additional padding $\mathfrak{h}$ and blindrotate it again. Then, we have that

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}_+) &= \mathsf{Phase}(\mathsf{acc}_{\mathsf{BR}}) \cdot X^{\mathsf{Phase}(\mathsf{ct}_{\mathsf{boot}})} + \mathfrak{e}_-\\ &= (-1)^{(1-b)} \left((-1)^{(1-b)} \left\lfloor \frac{Q}{2t} \right\rceil \mathfrak{h} + \mathfrak{e}_{\mathsf{BR}} \right) \cdot X^{\frac{2N}{t}w + \bar{e}} + \mathfrak{e}_-\\ &= \left(\left\lfloor \frac{Q}{2t} \right\rceil \mathfrak{h} + (-1)^{(1-b)} \cdot \mathfrak{e}_{\mathsf{BR}} \right) \cdot X^{\frac{2N}{t}w + \bar{e}} + \mathfrak{e}_-\end{aligned}$$

where $\mathfrak{e}_{\mathsf{BR}} = \mathfrak{h} \sum_{i=0}^{\frac{t}{2}} \mathfrak{e}_{+,\frac{2N}{t}\cdot i} + \mathfrak{e}_{\mathsf{RLWE}}$, and $\mathfrak{e}_{\mathsf{RLWE}}, \mathfrak{e}_-$ are the error terms stemming LWE to RLWE key-switching and from blind-rotation respectively. Finally, we observe the phase of $\mathsf{acc}^{(j)}$, discarding noise terms for now:

$$\begin{aligned}\mathsf{Phase}(\mathsf{acc}^{(j)}) &= \mathsf{Phase}(\mathsf{acc}_+) \cdot (\mathsf{rotP}_1 + \mathsf{rotP}_0) + \mathsf{Phase}(\mathsf{acc}_-)(\mathsf{rotP}_1 - \mathsf{rotP}_0)\\ &= \left\lfloor \frac{Q}{2t} \right\rceil \cdot \mathfrak{h} \cdot X^{\frac{2N}{t}w + \bar{e}} \cdot \big(\mathsf{rotP}_1 + \mathsf{rotP}_0 + (-1)^{1-b}(\mathsf{rotP}_1 - \mathsf{rotP}_0)\big)\\ &= \left\lfloor \frac{Q}{2t} \right\rceil \cdot \mathfrak{h} \cdot X^{\frac{2N}{t}w + \bar{e}} \cdot \begin{cases} 2 \cdot \mathsf{rotP}_0 & \text{if } b = 0\\ 2 \cdot \mathsf{rotP}_1 & \text{if } b = 1\end{cases}\\ &= \left\lfloor \frac{Q}{t} \right\rceil \cdot \mathfrak{h} \cdot X^{\frac{2N}{t}w + \bar{e}} \cdot \mathsf{rotP}_b\end{aligned}$$

Correctness again follows from the fact that $\mathfrak{h} \cdot \mathsf{rotP}_b$ is a polynomial encoding the values of F in chunks of size $\frac{2N}{t}$ and in descending order of inputs with inputs such that their most significant bit is equal to b, combined with the fact that $1 \leq \bar{e} < \frac{2N}{t}$.

We determine the variance of $\mathsf{acc}^{(j)}$ as for Algorithm 3. Then, noting that the results of $\mathsf{rotP}_0 \pm \mathsf{rotP}_1$ can be taken modulo t and that at most $\frac{t}{2}$ coefficients will be nonzero, it follows that

$$\begin{aligned}
\mathsf{Var}(\mathsf{Phase}(\mathsf{acc}^{(j)}) &\leq \mathsf{Var}(\mathsf{Phase}(\mathsf{acc}_{+}) \cdot (\mathsf{rotP}_0 + \mathsf{rotP}_1)) \\
&\quad + \mathsf{Var}(\mathsf{Phase}(\mathsf{acc}_{-}) \cdot (\mathsf{rotP}_0 - \mathsf{rotP}_1)) \\
&\leq \mathcal{B}_{\mathsf{BR}} \cdot \frac{t^3}{2} + \left(\left(\frac{t}{2} + 1\right)\mathcal{B}_{\mathsf{BR}} + \mathcal{B}_{\mathsf{pK}}\right) \cdot \frac{t^3}{2} \\
&\leq \left(t^3 + \frac{t^4}{4}\right)\mathcal{B}_{\mathsf{BR}} + \frac{t^3}{2}\mathcal{B}_{\mathsf{pK}}
\end{aligned}$$

and therefore

$$\mathcal{B}_{\mathsf{out}} \leq \left(t^3 + \frac{t^4}{4}\right)\mathcal{B}_{\mathsf{BR}} + \frac{t^3}{2}\mathcal{B}_{\mathsf{pK}}$$

□

3.3 The full cryptosystem.

For completeness, we briefly describe how the algorithms fit into the FHE definition described in Subsection 2.4.

Setup: We choose the modulus Q, a power-of-two dimension N of the ring $\mathcal{R}_Q$ and LWE dimension $n \in \mathbb{N}$. Then we choose $\mathfrak{s} \in \mathcal{R}_Q$ for the RLWE key, set $\mathbf{s}' \leftarrow \mathsf{KeyExt}(\mathfrak{s})$, and $\mathbf{s} \in \{0,1\}^n$ for the LWE key. Choose the radices $\mathsf{L}_{\mathsf{brK}}, \mathsf{L}_{\mathsf{ksK}}, \mathsf{L}_{\mathsf{pK}} \in \mathbb{N}$ and the Gaussian parameters for the noise σ, σ_{ksK}, σ_{brK}, $\sigma_{\mathsf{pK}} > 0$. Run $\mathsf{brK} \leftarrow \mathsf{BRKeyGen}(\sigma_{\mathsf{brK}}, \mathfrak{s}, \mathbf{s})$, $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\sigma_{\mathsf{ksK}}, \mathbf{s}, \mathbf{s}')$, and $\mathsf{pK} \leftarrow \mathsf{RLWEKeySwitchSetup}(\mathbf{s}', \mathfrak{s}, \mathsf{L}_{\mathsf{pK}}, \sigma_{\mathsf{pK}})$. Finally, set the evaluation key $\mathsf{ek} = (\mathsf{brK}, \mathsf{ksK}, \mathsf{pK})$ and the secret key $\mathsf{sk} = (\mathfrak{s}, \mathbf{s}', \mathbf{s})$.

Encryption: To encrypt a message $m' \in \mathbb{Z}_t$ we compute $\mathbf{c} = \mathsf{LWE}(\mathbf{s}', m) \in \mathbb{Z}_Q^{N+1}$ with standard deviation σ, where $m = \frac{Q}{t} \cdot m' \in \mathbb{Z}_Q$.

Eval: We can represent homomorphic computation as a circuit with gates of the form $f(b + \sum_{i=1}^{k} x_i \cdot a_i \in \mathbb{Z}_t) \in \mathbb{Z}_t$ where the a_i's and b are scalars known by the evaluator and the x_i's are the encrypted plaintexts. We compute the affine function using the additive homomorphism of the LWE samples, and the function $f : \mathbb{Z}_t \mapsto \mathbb{Z}_t$ by applying our bootstrapping algorithms from Section 3.

Decryption: Do decrypt an LWE sample $\mathbf{c}_{\mathsf{out}} = [\mathbf{a}_{\mathsf{out}}, b_{\mathsf{out}}]$ we run $\mathsf{Phase}(\mathbf{c}_{\mathsf{out}}) = \mathbf{c}_{\mathsf{out}}^{\top}[1, -\mathbf{s}] = b - \mathbf{a}_{\mathsf{out}}^{\top}\mathbf{s} = \frac{Q}{t}m'_{\mathsf{out}} + e \in \mathbb{Z}_t$, and round the result $\left\lceil \frac{t}{Q}\left(\frac{Q}{t}m'_{\mathsf{out}} + e\right) \right\rfloor = m'_{\mathsf{out}}$ if $|e| \leq \frac{Q}{2t}$.

4 Evaluation

In this section, we evaluate our bootstrapping algorithms. We give a theoretical comparison to other related works in Subsection 4.1 and give concrete parameters and timings in Subsection 4.2.

4.1 Theoretical Comparison to Related Works

In this section, we give a theoretical comparison to recent works that describe algorithms to approach functional bootstrapping namely [CLOT21, LMP22, KS23, CBSZ23, MHW+24], but primarily focus on the advantages of our methods over the functional bootstrapping algorithm introduced in [LMP22] and [CBSZ23]. For a comparison between [LMP22] and [CLOT21] & [KS23] we refer to Section 8.1 in [LMP22].

Table 3: Bounds on variances of different procedures employed. $\mathsf{ct} = \mathsf{LWE}(\mathbf{s}, \cdot)$ is a LWE sample of noise variance $\mathcal{B}_{\mathsf{ct}}$

Operation	Output Variance
$\mathsf{BlindRotate}(\mathsf{acc}, \mathsf{ct})$ [CGGI16]	$\mathcal{B}_{\mathsf{acc}} + 2 \cdot n \cdot N \cdot \log_{L_{\mathsf{brK}}}(Q) \cdot \frac{L_{\mathsf{brK}}^2}{12} \cdot \sigma_{\mathsf{brK}}^2$
$\mathsf{RLWEKeySwitch}_{\mathbf{s}' \to \mathfrak{s}}(\mathsf{ct}, \mathfrak{h})$ [CDKS21]	$\mathcal{B}_{\mathsf{ct}} + \lvert\lvert\mathfrak{h}\rvert\rvert_0 \cdot N^3 \cdot \log_{L_{\mathsf{pK}}}(Q) \cdot \frac{L_{\mathsf{pK}}^2}{12} \cdot \sigma_{\mathsf{pK}}^2$
$\mathsf{KeySwitch}_{\mathbf{s}' \to \mathbf{s}}(\mathsf{ct})$	$\mathcal{B}_{\mathsf{ct}} + \dim(\mathbf{s}') \cdot \log_{L_{\mathsf{ksK}}}(Q) \cdot \frac{L_{\mathsf{ksK}}^2}{12} \cdot \sigma_{\mathsf{ksK}}^2$
$\mathsf{ModSwitch}_{Q_0 \to Q_1}(\mathsf{ct})$	$\left(\frac{Q_1}{Q_0}\right)^2 \mathcal{B}_{\mathsf{ct}} + \frac{\lvert\lvert\mathbf{s}\rvert\rvert_0 \cdot \mathsf{Var}(\mathbf{s})}{4}$

We recall bounds on the noise variance introduced by the primitives we rely on in Table 3. We leverage the CGGI blind-rotation algorithm as in the original TFHE scheme [CGGI16]. Hence, we use binary LWE keys with entries in $\{0, 1\}$ and ternary keys for RLWE ciphertexts with coefficients in $\{-1, 0, 1\}$. We use the generic LWE-LWE key-switching algorithm (see, e.g., [KS23]), whereas the LWE to RLWE key-switching step, we rely on the k-LWE to RLWE packing algorithm from [CDKS21].

We give an overview of our comparison in Table 4[1]. In the case of FDFB-Compress [MHW+24], the ciphertext modulus for both blind-rotations is technically $2N$, but the entire plaintext space is compressed into $[0, N)$ after the initial blind-rotation. Hence, we effectively only utilize a modulus of N during the second blind rotation, which requires the same bounds on the error as if the modulus were equal to N.

We observe that for a single-function evaluation, both our methods and [LMP22] require exactly two blind-rotations. However, Algorithm 4 enables an unlimited amount of functions to be evaluated without increasing this number. Despite introducing a larger noise variance compared to Algorithm 3 or [LMP22], the increase can be compensated by adapting the parameter set, which may degrade the performance, but this can be disregarded if the number of functions is significant.

Table 4 and Table 3 suggest that both our algorithms introduce a significantly higher output variance. However, in practice, the situation is less dramatic. We note that in the case of [LMP22] we are a setting in which the LWE modulus is restricted to N, which can be attributed to how the algorithm works. At a high level, a ciphertext ct under modulus N is treated as if the modulus was $2N$, which introduces a random, most significant bit in its phase. By combining ct and the result of a blind-rotation of ct, this bit is cleared. Consequently, one obtains an LWE sample with fixed most-significant bit and therefore, we can entirely avoid the issue of negacyclity in a subsequent blind rotation. However, the price we must pay is that we only use half of the available plaintext, that is N instead of $2N$, whereas Algorithm 3 and Algorithm 4 have no such restrictions. Therefore, in a setting in which our algorithms and [LMP22] target the same plaintext space, we will be able to use a polynomial dimension that is half as large. Relying on smaller values of N affects not only the noise growth of blind rotation, but also leads to increased performance as the primary bottleneck lies with number-theoretic-transforms (NTT) required for polynomial multiplication which have a complexity of $O(N \log(N))$. We note that FDFB-CancelSign [MHW+24] uses similar concepts and therefore, the aforementioned points also apply to

[1]Partially adapted from [LMP22]

Table 4: Comparison to related works. L' denotes a basis. The formulas for the output noise of [KS23],[CLOT21],[LMP22] were taken from [LMP22]

Work	Output Variance	#Blind-rotations	q
[KS23]	$O(\sqrt{NL'} \cdot \log_{L'}(Q)) \cdot \mathcal{B}_{\mathsf{BR}}$	$\log_{L'}(Q) + 1$	$2N$
[CLOT21]	$O(N \cdot t) \cdot \mathcal{B}_{\mathsf{BR}}$	2	N
[LMP22]	$\mathcal{B}_{\mathsf{BR}}$	2	N
[LMP22] - MV	$Nt^2\mathcal{B}_{\mathsf{BR}}$	2	N
[CBSZ23] (ComBo)	$2\mathcal{B}_{\mathsf{BR}}$	4	$2N$
[CBSZ23] (ComBoMV)	$2\mathcal{B}_{\mathsf{BR}}$	3	$2N$
[MHW$^+$24] (FDFB-Compress)	$\mathcal{B}_{\mathsf{BR}}$	2	N
[MHW$^+$24] (FDFB-CancelSign)	$\mathcal{B}_{\mathsf{BR}}$	2	N
Algorithm 3	$\mathcal{B}_{\mathsf{BR}} \cdot \left(\frac{t^3}{2} + 1\right) + \mathcal{B}_{\mathsf{pK}} \cdot \frac{t^3}{2}$	2	$2N$
Algorithm 4	$\left(t^3 + \frac{t^4}{4}\right)\mathcal{B}_{\mathsf{BR}} + \frac{t^3}{2}\mathcal{B}_{\mathsf{pK}}$	2	$2N$

this method, and we do not specifically compare our methods to [MHW$^+$24].

We are not aware of prior work applying the multi-value bootstrapping technique to the method of [LMP22]. Nevertheless, we note that the method can be easily applied to [LMP22] at the cost of decreasing the decomposition factors. We give our parameter set at Table 6 for the multi-value version of [LMP22]. Similarly to Algorithm 4, the computational cost increases only by a negligible amount, however, at the cost of amplifying the error induced through blind-rotation by a factor of $N \cdot t^2$. We include this version of the algorithm in Table 4.

Clet et al. introduce two algorithms, ComBo and ComBoMV [CBSZ23], that leverage the full modulus $2N$. At a high level, the authors first decompose a given function into a pair of (quasi-) even and (quasi-) odd functions, both of which can be evaluated using two blind rotations. Then, we obtain the desired value by combining the results of the function evaluation. The default method ComBo therefore requires 4 blind-rotations but drops to 3 if we rely on the amortization method of [CIM19].

Finally, we note that our algorithms will require additional key material for the LWE to RLWE key-switch operation, which requires storing $2N \log_2(N) \log_{L_{\mathsf{pK}}}(Q)$ integers of $\log(Q)$ bits as we leverage [CDKS21]. As the size may grow quickly for larger values of N, we point out that it is possible to *entirely* eliminate the need for additional key material. In [LMK$^+$23], the authors describe a blind-rotation algorithm that relies on the homomorphic evaluation of automorphisms of the ring $\mathcal{R}$. The same techniques had been previously used in [CDKS21] to perform the packing of multiple LWE samples into a single RLWE ciphertext. Therefore, by carefully generating the necessary key material required by [LMK$^+$23], we will be in a position to reuse the same keys to perform the LWE-RLWE key-switch without using dedicated keys.

4.2 Parameters and Bootstrap Timings

In this section, we give a practical evaluation of our algorithm.

We implement our algorithm in the FHE-Deck library[2] and perform our experiments on an Intel i7 14700KF processor using 64GiB RAM with no parallelization. We compare our algorithms to [LMP22] and [CBSZ23] in practice, as they both represent the state of the art for the case $q = N$ and $q = 2N$ respectively. For completeness, we note that we did not re-implement the algorithms described [CBSZ23], but we measured the time for the necessary amount of blind rotations using the selected parameters.

[2] https://github.com/FHE-Deck/fhe-deck-core

Table 5: Parameter sets used, λ denotes the security level.

Set	λ	n	N	$\log_2(Q)$	$\log_2(Q_{\mathsf{ksK}})$	σ_{brK}	σ_{ksK}	L_{ksK}	L_{brK}	L_{pK}
SET-I	140	900	2^{11}	51	30	2^4	2^{12}	2^3	2^{17}	2^{10}
SET-II	130			52				2^2	2^{13}	

Table 6: Parameter sets used for [LMP22] and [CBSZ23] instantiation, λ denotes the security level.

Set	λ	n	N	$\log_2(Q)$	$\log_2(Q_{\mathsf{ksK}})$	σ	σ_{ksK}	L_{ksK}	L_{brK}
LMP-I	128	900	2^{12}	48	20	3.19	3.19	2^3	2^{16}
LMP-II									2^{12}
COMBO-I	128	900	2^{11}	48	20	3.19	3.19	2^4	2^{16}

We give an overview of the parameters chosen for Algorithm 3 and Algorithm 4 in Table 5 and the parameters for [LMP22] and[CBSZ23] in Table 6. We selected the sets in order to target a failure rate of 2^{-60} for a plaintext space of $t = 2^5$. In Table 7, we show the failure probabilities for each parameter set for plaintext spaces $t \in \{2^4, 2^5, 2^6\}$ and give the number of subsequent evaluations necessary for a failure to occur with a probability exceeding 0.5.

We computed the failure probabilities as follows. Let $\mathcal{B}_{\mathsf{boot}}$ be the output variance of ciphertext obtained by a bootstrap algorithm as given in Table 4. In order to perform blind rotation using such a ciphertext, we modulus-switch it to Q_{ksK}, key-switch it to a secret key of dimension equal to n, and modulus-switch it to a modulus $q = N$ in case of [LMP22] or $q = 2N$ otherwise. Leveraging Table 3, we determine the noise variance of the result:

$$\mathcal{B} \leq \left(\frac{q}{Q}\right)^2 \mathcal{B}_{\mathsf{boot}} + \left(\frac{q}{Q_{\mathsf{ksK}}}\right)^2 \mathcal{B}_0 + \mathcal{B}_1$$

where

$$\mathcal{B}_0 \leq N \cdot \log_{L_{\mathsf{ksK}}}(Q_{\mathsf{ksK}}) \cdot \frac{L_{\mathsf{ksK}}^2}{12} \cdot \sigma_{\mathsf{ksK}}^2 + \frac{||\mathsf{s}'||_0 \cdot \mathsf{Var}(\mathsf{s}')}{4}$$
$$\mathcal{B}_1 \leq \frac{||\mathsf{s}||_0 \cdot \mathsf{Var}(\mathsf{s})}{4}$$

The error distribution is commonly modeled as a (discrete) Gaussian with mean 0 and variance $\mathcal{B}$. Hence, we may use the error function to determine the probability that the ciphertext modulo q of dimension n cannot be decrypted, which equals the probability that the magnitude of the noise term e exceeds $\frac{q}{2t}$ in absolute value:

$$\begin{aligned}\mathsf{P}[\mathsf{Fail}] &= 1 - \mathsf{P}\left[-\frac{q}{2t} < |e| < \frac{q}{2t}\right] \\ &= 1 - \int_{-q/2t}^{q/2t} \frac{1}{\sqrt{2\pi\mathcal{B}}} \exp\left(-\frac{t^2}{2\mathcal{B}}\right) dt \\ &= 1 - \mathsf{erf}\left(\frac{q}{2 \cdot t \cdot \sqrt{2\mathcal{B}}}\right)\end{aligned}$$

Table 8 gives an overview of all obtained timings, and we briefly discuss the results. Algorithm 3 yields the fastest evaluation, outperforming ComBoMV [CBSZ23] by 15%

Table 7: Failure probabilities for each parameter set. $\mathsf{P}[\mathsf{Fail}]$ corresponds to the probability that the bootstrapping procedure outputs a result differing from the expected one. $n_{\mathsf{eval}}[0.5]$ denotes the number of evaluation necessary for a failure to occur with a probability exceeding 0.5

Set	$\mathsf{P}[\mathsf{Fail}]$			$n_{\mathsf{eval}}[0.5]$		
	t = 16	t = 32	t = 64	t = 16	t = 32	t = 64
SET-I	2^{-256}	2^{-65}	2^{-16}	$8.03 \cdot 10^{76}$	$2.56 \cdot 10^{19}$	$4.54 \cdot 10^{4}$
SET-II	2^{-319}	2^{-66}	2^{-18}	$7.40 \cdot 10^{95}$	$5.11 \cdot 10^{19}$	$1.82 \cdot 10^{5}$
LMP-I	2^{-233}	2^{-60}	2^{-16}	$9.57 \cdot 10^{69}$	$7.99 \cdot 10^{17}$	$4.54 \cdot 10^{4}$
LMP-II	2^{-232}	2^{-60}	2^{-16}	$4.78 \cdot 10^{69}$	$7.99 \cdot 10^{17}$	$4.54 \cdot 10^{4}$
COMBO-I	2^{-180}	2^{-53}	2^{-14}	$1.06 \cdot 10^{54}$	$6.24 \cdot 10^{15}$	$1.14 \cdot 10^{4}$

Table 8: Bootstrap timings in milliseconds.

Work	Algorithm	Parameter Set	Time in ms.	ek Size [MB]	Multi-Value
Ours	Algorithm 3	SET-I	596	188	×
Ours	Algorithm 4	SET-II	721	217	✓
[CBSZ23]	ComBo	COMBO-I	925	88	×
[CBSZ23]	ComBoMV	COMBO-I	702	88	×
[LMP22]		LMP-I	876	176	×
[LMP22] - Amort		LMP-II	1050	236	✓

and [LMP22] by 31%. Algorithm 4 remains faster than [LMP22] and its multi-value counterpart by a margin of 17% and 31% respectively, but is slightly slower than ComBoMV. Nevertheless, we believe that our algorithm remains superior as it will be possible to amortize the runtime over the computation of several functions, which, in the case of ComBoMV, requires additional blind rotations. Note that the size of the public keys may differ heavily on the representation of the data structure. Here, we calculate the size of the public keys by counting the number of integers in ring elements. For each integer, we reserve 8 bytes. Furthermore, we assume the folklore optimization where one can compute all the uniform random parts of a (R)LWE samples in the evaluation keys from a compact seed and a secure pseudo-random number function. We apply this optimization to all parameter sets.

5 Conclusion

In this work, we introduced two novel algorithms that resolve the issue of negacyclity. Our first algorithm evaluates an arbitrary function on a ciphertext with a small noise variance, whereas the second algorithm enables us to evaluate an unlimited amount of functions at little to no computational overhead but with a larger output variance. Our novel algorithms offer theoretical advantages that manifest in practice through a faster bootstrapping procedure and algorithms that are simple to implement. We leave it to the future work to explore further potential efficiency gains, which may be obtained by leveraging blind-rotations methods that rely on the NTRU assumption such as [BIP$^+$22] or NTRU-ν-um [Klu22].

Acknowledgments

We thank Aysajan Abidin for proofreading the manuscript and for his helpful comments.

This work was partly supported by the Flemish Government through FWO SBO project MOZAIK S003321N. It was also funded in part by CyberSecurity Research Flanders with reference number VR20192203.

References

[ADDG23] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious PRFs from shallow PRFs and FHE. Cryptology ePrint Archive, Report 2023/232, 2023. https://eprint.iacr.org/2023/232.

[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.

[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.

[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.

[BIP$^+$22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 188–215. Springer, Heidelberg, December 2022.

[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.

[CBSZ23] Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, and Martin Zuber. ComBo: A novel functional bootstrapping method for efficient evaluation of nonlinear functions in the encrypted domain. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 317–343. Springer Nature, July 2023.

[CDKS21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Heidelberg, June 2021.

[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.

[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.

[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. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Heidelberg, December 2021.

[DJL$^+$24] Amit Deo, Marc Joye, Benoit Libert, Benjamin R. Curtis, and Mayeul de Bellabre. Homomorphic evaluation of lwr-based prfs and application to transciphering. Cryptology ePrint Archive, Paper 2024/665, 2024. https://eprint.iacr.org/2024/665.

[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.

[fhe23] Fhe-deck. https://github.com/FHE-Deck, September 2023.

[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.

[JW22] Marc Joye and Michael Walter. Liberating tfhe: Programmable bootstrapping with general quotient polynomials. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 1–11, New York, NY, USA, 2022. Association for Computing Machinery.

[Klu22] Kamil Kluczniak. NTRU-v-um: Secure fully homomorphic encryption from NTRU with small modulus. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1783–1797. ACM Press, November 2022.

[KS23] Kamil Kluczniak and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR TCHES*, 2023(1):501–537, 2023.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 227–256. Springer, Heidelberg, April 2023.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 130–160. Springer, Heidelberg, December 2022.

[MHW+24] Shihe Ma, Tairong Huang, Anyu Wang, Qixian Zhou, and Xiaoyun Wang. Fast and accurate: Efficient full-domain functional bootstrap and digit decomposition for homomorphic computation. *IACR TCHES*, 2024(1):592–616, 2024.

[MKG23] Johannes Mono, Kamil Kluczniak, and Tim Güneysu. Improved circuit synthesis with amortized bootstrapping for fhew-like schemes. Cryptology ePrint Archive, Paper 2023/1223, 2023. https://eprint.iacr.org/2023/1223.

[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.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg10] Oded Regev. On the complexity of lattice problems with polynomial approximation factors. ISC, pages 475–496. Springer, Heidelberg, 2010.

[WWL+24] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. Circuit bootstrapping: Faster and smaller. Cryptology ePrint Archive, Paper 2024/323, 2024. https://eprint.iacr.org/2024/323.

[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.