Circuit Privacy for FHEW/TFHE-Style Fully Homomorphic Encryption in Practice

Kamil Kluczniak

CISPA Helmholtz Center for Information Security
kamil.kluczniak@cispa.de

Abstract. A fully homomorphic encryption (FHE) scheme allows a client to encrypt and delegate its data to a server that performs computation on the encrypted data that the client can then decrypt. While FHE gives confidentiality to clients' data, it does not protect the server's input and computation. Nevertheless, FHE schemes are still helpful in building delegation protocols that reduce communication complexity, as FHE ciphertext's size is independent of the size of the computation performed on them.
We can further extend FHE by a property called circuit privacy, which guarantees that the result of computing on ciphertexts reveals no information on the computed function and the inputs of the server. Thereby, circuit private FHE gives rise to round optimal and communication efficient secure two-party computation protocols. Unfortunately, despite significant efforts and much work put into the efficiency and practical implementations of FHE schemes, very little has been done to provide useful and practical FHE supporting circuit privacy. In this work, we address this gap and design the first randomized bootstrapping algorithm whose single invocation sanitizes a ciphertext and, consequently, serves as a tool to provide circuit privacy. We give an extensive analysis, propose parameters, and provide a C++ implementation of our scheme. Our bootstrapping can sanitize a ciphertext to achieve circuit privacy at an 80-bit statistical security level in 1.4 seconds. In addition, we can perform non-sanitized bootstrapping in around 0.14 seconds on a laptop without additional public keys. Crucially, we do not need to increase the parameters significantly to perform computation before or after the sanitization takes place. For comparison's sake, we revisit the Ducas-Stehlé washing machine method. In particular, we give a tight analysis, estimate efficiency, review old and provide new parameters.

1 Introduction

Fully homomorphic encryption (FHE) is an encryption scheme that allows performing arbitrary computation on encrypted data. A client encrypts a message m and sends the ciphertext to a server which, given a function F, returns another ciphertext that decrypts to $F(m)$. The concept of FHE was first introduced by Rivest, and Dertouzos [60], and the first theoretical realization of that concept is due to Gentry [35].

A critical property for FHE is circuit privacy. Roughly speaking, the ciphertext that is the product of the server computing a function F on encrypted data should not reveal any information about F except that the ciphertext decrypts to $F(m)$. To prove circuit privacy, we need to show a simulator that, on input $F(m)$ and a public key, outputs a fresh encryption of $F(m)$, which is indistinguishable from the servers' computed ciphertext. In particular, the distribution of an evaluated ciphertext should be close to or the same as the distribution of a fresh encryption.

We can easily see that circuit private FHE gives us semi-honest two-party computation with optimal communication. Namely, we only need one round of communication. The first message can be reused, and the communication complexity is independent of the size of the computation. Furthermore, we can reuse the ciphertexts output from the evaluation process and keep computing on them. Since circuit private FHE gives us two-party computation, all applications for two-party computation protocols apply here as well. Among other these are private set intersection [43, 54, 20], neural network inference [28, 17, 51, 45, 46, 12, 3, 18, 59, 10] or analysis on genomic data [47, 48, 10]. Note that circuit privacy is not always needed. Without circuit privacy FHE reduces to secure delegation. For example, in (single-server) private information retrieval we are only interested in protecting the user's query, but not in the confidentiality of a potentially large database of the server. On the other hand, we believe that for neural network inference, as an example, confidentiality of the neural network is essential. In contrast to PIR, it is difficult to make an argument for compressing the communication costs, as current FHE schemes require sending public keys that are order of magnitude larger than the size of deep neural networks.

Surprisingly, despite over a decade of advances in constructing scalable fully homomorphic encryption schemes [36, 15, 14, 37, 7, 40, 16, 8, 41, 29, 21, 19, 22, 42], and numerous implementations [1, 23, 24, 2] there is very little constructions and nearly no implementation that we are aware of that natively provide circuit privacy.

Current approaches to Circuit Privacy. In this paper, we are interested in fully homomorphic encryption as in [35]. Namely, ciphertexts do not grow with the size of computation, and evaluation results are reusable. A trivial way to re-randomize a ciphertext is to create a fresh encryption of zero using the public key and add it to the ciphertext resulting from the computation. Unfortunately, such an approach is insufficient to provide circuit privacy in current FHE schemes because all secure FHE schemes we know are based on noisy encryptions. This noise depends on the computed circuit and is the main obstacle to overcome when re-randomizing (or sanitizing) a ciphertext to provide circuit privacy.

Below we summarize current approaches.

Noise Flooding: Introduced in Gentry's thesis [34] requires adding a fresh ciphertext of zero and a super-polynomially larger noise term to the sanitized ciphertext. Unfortunately, in practice, this additional noise term is substantially large and would require us to choose very big parameters. We note that it is required to take the noise super-polynomially larger than the noise in the sani-

tized ciphertext. Hence, if the noise in this ciphertext is already large due to some previous computation, then the the magnitude of the additional noise must be chosen accordingly. Nevertheless, the method has found some applications in leveled homomorphic computation [20] where we do not bootstrap the ciphertexts and can tolerate larger parameters.

Ducas-Stehlé Washing Machine: Introduced by Ducas and Stehlé [30], requires to run a sequence of re-randomization steps (flooding cycles), each with a smaller noise flooding error, followed by invocations of a bootstrapping algorithm. The paper only roughly estimates the number of times re-randomization and bootstrapping must be invoked. However, as the authors admit, the estimates should be taken with great caution and defer a concrete analysis to future work. For example, they suggest running the FHEW [29] bootstrapping algorithm between 8 and 16 times. It is not entirely clear what security level they are able to achieve and whether the parameter set of the FHEW algorithm proposed at the time satisfies the given correctness constraints. To the best of our knowledge, no concrete analysis or implementations have been done so far.

Secure Two-Party Computation: A few works [38, 25] proposed to use garbled circuit-based techniques to provide circuit privacy. Gentry, Halevi, Vaikuntanathan [38] give a non-compact homomorphic encryption scheme that can be thought of as a re-randomizable version of garbled circuits. We are not aware of the scheme's implementation; however, we note that the scheme is not compact. In particular, the communication complexity is linear in the size of the computed circuit. Finally, Chongchitmate and Ostrovsky [25] propose to use garbled circuits to compute the decryption step.

Rerandomizing Computation: A promising result was given by Bourse, Pino, Minelli and Wee [13]. Their approach exploits properties of the Gentry, Sahai, Waters (GSW) cryptosystem [40]. Specifically, when multiplying GSW ciphertexts, we use a randomized version of gadget decomposition instead of a deterministic one. In [13] the authors show that when gadget decomposition is implemented via Gaussian sampling [56, 32] with appropriate parameters, then we can build an FHE scheme for circuits of depth logarithmic in the number of inputs. The results are mostly asymptotic, without concrete parameter proposals nor implementation.

1.1 Our Contribution.

We design a randomized FHEW/TFHE-style [29, 21] bootstrapping algorithm that can sanitize a given ciphertext. In contrast to the Ducas-Stehlé washing machine method [30], which we shortly refer to as DS-WM, we need to run our bootstrapping algorithm only once. Our results solve an open problem posted in [13], in that we use their randomization concept in an FHEW-style bootstrapping scheme. We note that porting the ideas from [13] to the ring setting is non-trivial since there are no analogs of the leftover hash lemma [27] and Gaussian leftover hash lemma [5, 4, 13] for the ring setting. Moreover, we can argue that designing a "leaky" analog of the regularity lemmas [61, 53] as in [26], may result in practically inefficient bootstrapping. We show how to overcome both problems

using simple techniques in the right places by exploiting structural properties of FHEW/TFHE [29, 21] instantiated over the ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ where N is a power of two. Along the way, we generalize the technical lemmas from [13] to support any modulus $Q \in \mathbb{N}$, instead of moduli of the form $Q = \mathsf{L}^\ell$ for some $\mathsf{L}, \ell \in \mathbb{N}$.

We compare our method with DS-WM that is the most competitive. While [30] give a heuristic instantiation based on FHEW [29], they left a serious analysis as an open problem. We resolve the problem and give a tight error analysis, and provide scripts that automate noise and security estimations of our randomized bootstrapping and DS-WM. We show that the parameters proposed in [30] cannot be circuit private due to correctness issues. In general, we show that instantiations over a ring of dimension 2^{10}, or smaller, cannot give more than 30-bits of statistical security. Note that many efficient bootstrapping schemes [29, 21, 22] are instantiated over rings of dimension 2^{10}.

Finally, we give an efficient C++ implementation[1] of our bootstrapping algorithms. To the best of our knowledge, this is the first practical realization of a circuit private FHEW/TFHE-style FHE scheme. Our implementation supports number theoretic transform-based (NTT) and fast Fourier transform-based (FFT) multiplication of ring elements. Due to our versatile implementation, we can experiment with different moduli choices that lead to different algorithm variants. In particular, we can instantiate different Gaussian samplers that are optimized toward a specific choice of modulus. We choose different parameters targeting the different representations. Nevertheless, we identify some drawbacks to the FFT-based implementation due to the relatively low precision of the floating point arithmetic.

Nevertheless, our algorithm scan sanitizes a ciphertext in 1.4 seconds and 1.2 seconds for the NTT and FFT-based implementation, respectively. Furthermore, with no additional public key, we can compute standard FHEW/TFHE-style bootstrapping in around 0.14 and 0.27 seconds for NTT and FFT, respectively. We show that the DS-WM method has roughly 2.46× and 5.22× slower sanitization for NNT and FFT, respectively, for parameters that have the same key sizes, non-sanitized (deterministic) computation time, and correctness level.

1.2 Our Techniques.

In this section, we discuss the most important issues and problems. We only focus on the multiplication of GSW ciphertexts with integers in $\mathbb{Z}_Q$, and we omit giving an overview of FHEW/TFHE schemes so that we can showcase ideas most critical for circuit privacy. At a high level, we use the ideas from [13]. Remind that [13] gives a randomized version of the GSW cryptosystem [40], which is the core algorithm underlying the bootstrapping algorithms from [8, 29].

Brief Overview of [13]. First we define a gadget vector $\mathbf{g} = [1, 2, \ldots, 2^\ell]$ and the matrix $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n \in \mathbb{Z}_Q^{n \times \ell n}$, where $\mathbf{I}_n$ is the n dimensional identity matrix.

[1] Available at `https://github.com/FHE-Deck`.

A GSW encryption of a message $m \in \mathbb{Z}_Q$ is given as

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top \end{bmatrix} + m \cdot \mathbf{G},$$

where $\mathbf{s} \in \mathbb{Z}_Q^n$ is the secret key, $\mathbf{A} \in \mathbb{Z}_Q^{(n-1)\times \ell n}$ is public, and $\mathbf{e} \in \mathbb{Z}_Q^{\ell n}$ is a noise vector whose entries are from the discrete Gaussian distribution.

Define a randomized gadget decomposition algorithm $\mathbf{X} \leftarrow \mathsf{G}_{\mathsf{rand}}^{-1}(a \cdot \mathbf{G})$, where $a \in \mathbb{Z}_Q$ and the matrix $\mathbf{X} \in \mathbb{Z}_Q^{\ell n \times \ell n}$ is from the discrete Gaussian distribution such that $\mathbf{G} \cdot \mathbf{X} = a \cdot \mathbf{G}$. We can use $\mathsf{G}_{\mathsf{rand}}^{-1}$ to multiply the GSW ciphertext $\mathbf{C} \in \mathbb{Z}_Q^{n \times \ell n}$ by a and randomize the outcome as follows.

$$\mathbf{C} \cdot \mathbf{X} + \begin{bmatrix} \mathbf{0} \\ \mathbf{y}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{A} \cdot \mathbf{X} \\ \mathbf{s}^\top \mathbf{A} \cdot \mathbf{X} + \mathbf{e}^\top \mathbf{X} + \mathbf{y}^\top \end{bmatrix} + m \cdot a \cdot \mathbf{G},$$

where $\mathbf{y} \in \mathbb{Z}_Q^{\ell n}$ is chosen from the discrete Gaussian distribution. The main idea and technical contribution in [13] is to show that such a product already gives us a ciphertext that is statistically independent of the input ciphertext. At the heart of their proof is the core randomization lemma that states that a tuple $(\mathbf{A} \cdot \mathbf{X}, \mathbf{e}^\top \mathbf{X} + \mathbf{y}^\top)$ is statistically indistinguishable from $(\bar{\mathbf{A}}, \bar{\mathbf{e}}^\top)$, where $\bar{\mathbf{A}} \in \mathbb{Z}_Q^{(n-1)\times \ell n}$ is from the uniform distribution and $\bar{\mathbf{e}} \in \mathbb{Z}_Q^{\ell n}$ is an independent random variable from the discrete Gaussian distribution with a slightly higher standard deviation, given that $\mathbf{X}$ and $\mathbf{y}$ have sufficiently high standard deviations.

The first step to prove the core randomization lemma is to show that $\mathbf{A} \cdot \mathbf{X}$ is close to uniform from the generalized leftover hash lemma [27]. To this end, we need to analyze the entropy of $\mathbf{X}$ given $\mathbf{e}^\top \mathbf{X} + \mathbf{y}^\top$, and $\mathbf{e}$. To show that $\mathbf{e}^\top \mathbf{X} + \mathbf{y}^\top$ is close to an independent discrete Gaussian random variable, [13] use an adaptation of the Gaussian leftover hash lemma [5, 4].

In practice, FHE instantiated over LWE is quite slow. Hence we usually want to instantiate a scheme in the ring setting. We are not aware of any serious implementation in the LWE setting. In this paper, we draw our attention to FHEW/TFHE-style schemes [29, 21], which are based on an efficient bootstrapping algorithm that, on input an LWE ciphertext, output a refreshed LWE ciphertext with smaller noise.

Problems with translating [13] into the Ring Setting. At the heart of FHEW/TFHE-style bootstrapping algorithm sis an algorithm called blind rotation that outputs an RLWE ciphertext $\mathbf{c} = (\mathfrak{b}, \mathfrak{a}) \in \mathcal{R}_Q^2$ whose constant coefficient of the encrypted message encodes the decryption of an input ciphertext. We can show that the ciphertext can be represented as $(\mathbf{a}^\top \mathbf{x}, \mathbf{e}^\top \mathbf{x} + \mathfrak{y})$ as above but where $\mathfrak{y} \in \mathcal{R}_Q$, $\mathbf{a}, \mathbf{x}, \mathbf{e} \in \mathcal{R}_Q^m$ and elements in $\mathbf{x}$ have coefficients from the discrete Gaussian distribution. Remind that $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

If we want to follow the technique from [13], we would need to show that $\mathfrak{a} = \mathbf{a}^\top \mathbf{x}$ is close to uniform given $\mathbf{e}^\top \mathbf{x} + \mathfrak{y}$ and $\mathbf{e}$. Unfortunately, there is no analog of the leftover hash lemma lemma for rings like $\mathcal{R}_Q$. Consider the example where the j-th NTT coordinate of the elements in $\mathbf{x}$ and $\mathfrak{y}$ leaks. Clearly, $\mathbf{x}$ and $\mathfrak{y}$ may still have high entropy, but anyone can distinguish $\mathbf{a}^\top \mathbf{x}$ from uniform just

by looking at the j-th NTT coordinate. We may try to define a "leaky" version of the regularity lemma [53] as in [26]. But we argue that even if we would ignore the leak, the regularity lemma from [53] produces a practically inefficient (for our application) solution because it requires us to choose a small decomposition basis resulting in high ℓ and, consequently, in relatively slow (Ring) GSW products. Otherwise, we need to choose a high standard deviation σ_x of $\mathbf{x}$ resulting in larger parameters or lower correctness. Concretely, we must choose the standard deviation σ_x of $\mathbf{x}$ to be larger than $N \cdot Q^{1/\ell+2/N\ell}$ to achieve negligible security. The ℓ parameter is critical as it affects the most time-consuming operation in the bootstrapping scheme. Hence it is imperative to keep ℓ small in practical implementations. Another problem is that we do not have a ring analog of the Gaussian leftover hash lemma.

To bypass these problems, we exploit that in FHEW/TFHE-style schemes, we extract an LWE ciphertext from $\mathbf{c}$. In particular, observe that $(b', \mathbf{a}') \in \mathbb{Z}_Q^{N+1}$, where $b' = \mathfrak{b}[1] \in \mathbb{Z}_Q$ is $\mathfrak{b}$'s constant coefficient, and $\mathbf{a}' \in \mathbb{Z}_Q^N$ is such that $\mathbf{a}'[1] = \mathfrak{a}[1]$ and $\mathbf{a}'[i] = \mathfrak{a}[N-i]$ for $i = 0 \dots N-2$, is a correct LWE ciphertext with respect to the secret key $\mathbf{s}' = \mathfrak{s}$ (the coefficient vector of $\mathfrak{s}$) encrypting the constant coefficient of $\mathbf{c}$'s message. Note that we still cannot claim that $\mathbf{a}'$ is close to uniform, but what we can do is sample a fresh LWE ciphertext of 0, add it to $(b', \mathbf{a}')$ obtaining a ciphertext $(\bar{b}, \bar{\mathbf{a}})$ of the same message where $\bar{\mathbf{a}}$ is statistically close to uniform. Finally, we can show that the error term of $(b', \mathbf{a}')$ is already in the form required by the Gaussian leftover hash lemma [5, 4, 13]. To show this, we exploit the specific structure of the ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ where the product of two ring elements is a negacyclic convolution of two polynomials. Our analysis applies only to $\mathcal{R}_Q$, but on the other hand, FHEW/TFHE-style bootstrapping exploits the structure of $\mathcal{R}_Q$ for correctness.

2 Background and Notation

We denote as $\mathcal{R}_Q$ the ring of polynomials $\mathbb{Z}_Q[X]/(X^N+1)$ where N is a power of two. We only use Q and N in the context of the ring $\mathcal{R}_Q$. We denote vectors with bold lowercase letters, e.g., $\mathbf{v}$, and matrices with uppercase letters $\mathbf{V}$. We denote an 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 $[i]_{i=n}^m$ the vector $[n, \dots, m]$. We address the ith entry of a vector $\mathbf{v}$ by $\mathbf{v}[i]$. For matrices we address the ith row and jth column as $\mathbf{A}[i,j]$. Sometimes we view ring elements $\mathfrak{a} \in \mathcal{R}_Q$ as vectors of coefficients and we address the coefficients as vector coordinates. For a random variable $x \in \mathbb{Z}$ we denote as $\mathsf{Var}(x)$ the variance of x, as $\mathsf{stddev}(x)$ its standard deviation and as $\mathsf{E}(x)$ its expectation. For $a \in \mathcal{R}_Q$, we define $\mathsf{Var}(a)$, $\mathsf{stddev}(a)$ and $\mathsf{E}(a)$ to be the largest variance, standard deviation and expectation respectively among the coefficients of the polynomial a. By $\mathsf{Ha}(\mathbf{a})$ we denote the hamming weight of the vector $\mathbf{a}$, i.e., the number of of non-zero coordinates of $\mathbf{a}$. We represent numbers in $\mathbb{Z}_Q$ as integers in $[-Q/2, Q/2)$.

We say that an algorithm is **PPT** if it is a probabilistic polynomial-time algorithm. We denote any polynomial as $\mathsf{poly}(.)$. We denote as $\mathsf{negl}(\lambda)$ a negligible function in $\lambda \in \mathbb{N}$. That is, for any positive polynomial $\mathsf{poly}(.)$ there exists $c \in \mathbb{N}$ such that for all $\lambda \geq c$ we have $\mathsf{negl}(\lambda) \leq \frac{1}{\mathsf{poly}(\lambda)}$. Given two distributions X, Y over a finite domain D, their statistical distance is defined as $\Delta(X,Y) = \frac{1}{2}\sum_{v \in D} |X(v) - Y(v)|$. We say that two distributions are statistically close if their statistical distance is negligible.

Lattices. A m-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^m$. For an integer $k < m$ and a rank matrix $\mathbf{B} \in \mathbb{R}^{m \times k}$, $\Lambda(\mathbf{B}) = \left\{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^k\right\}$ is the lattice generated by the columns of $\mathbf{B}$. We denote $\Lambda_q^{\perp}(\mathbf{B}) = \left\{\mathbf{v} \in \mathbb{Z}^m : \mathbf{B}^{\top}\mathbf{v} = 0 \mod q\right\}$.

Gaussian distribution. For any $\sigma > 0$, the spherical Gaussian function with parameter σ is defined as $\rho_\sigma(\mathbf{x}) = \exp\left(\frac{-\pi||\mathbf{x}^2||}{\sigma^2}\right)$, for any $\mathbf{x} \in \mathbb{R}^m$. Given a lattice $\Lambda \subseteq \mathbb{R}^m$, a parameter $\sigma \in \mathbb{R}$ and a vector $\mathbf{c} \in \mathbb{R}^m$ the spherical Gaussian distribution with parameter σ and support $\Lambda + \mathbf{c}$ is defined as

$$\mathcal{D}_{\Lambda+\mathbf{c},\sigma}(\mathbf{x}) = \frac{\rho_\sigma(\mathbf{x})}{\rho_\sigma(\Lambda+\mathbf{c})}, \forall \mathbf{x} \in \Lambda + \mathbf{c}$$

where $\rho_\sigma(\Lambda+\mathbf{c})$ denotes $\sum_{\mathbf{x} \in \Lambda+\mathbf{c}} \rho_\sigma(\mathbf{x})$.

We write $x \leftarrow_{\$} \mathcal{D}_{\Lambda+\mathbf{c},\sigma}$ do denote that x is sampled from the discrete Gaussian distribution with support $\Lambda + \mathbf{c}$ and parameter σ. We write $\mathfrak{y} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}^N,\sigma}$ or $\mathbf{y} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}^n,\sigma}$ when sampling the coefficients of $\mathfrak{y} \in \mathcal{R}$ or components of $\mathbf{y} \in \mathbb{Z}^N$ from $\mathcal{D}_{\mathbb{Z},\sigma}$. For a set $\mathcal{S}$ we write $x \leftarrow_{\$} \mathcal{S}$ to denote the uniform distribution over $\mathcal{S}$ unless said otherwise. Throughout the paper we denote $C_{\delta,m} = \sqrt{\frac{\ln(2m(1+1/\delta))}{\pi}}$.

Learning With Errors. We recall the learning with errors assumption by Regev [58]. Our description is a generalized version due to Brakerski, Gentry, and Vaikuntanathan [14].

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

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

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

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

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

We denote a Learning With Errors (LWE) sample as $\mathsf{LWE}_{\sigma,n,Q}(\mathbf{s},\ \mathsf{m}) = \mathsf{GLWE}_{\sigma,n,1,Q}$, which is a special case of a GLWE sample where the ring is

$\mathbb{Z}_q[X]/(X+1)$. Similarly we denote a Ring-Learning with Errors (RLWE) sample as $\mathsf{RLWE}_\sigma(\mathbf{s},\mathsf{m}) = \mathsf{GLWE}_{\sigma,1,N,Q}$ which is the special case of an GLWE sample with $n=1$. For simplicity, we omit to state the modulus and ring dimension for RLWE samples because we always use $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ where N is a power of two. For LWE samples, we will be switching between different moduli and different dimensions; hence we will indicate the current modulus in the notation. Sometimes we use the notation $\mathbf{c} \in \mathsf{GLWE}_{\sigma,n,1,Q}(\mathbf{s},\ \mathsf{m})$ (resp. LWE and 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. We define the phase of $\mathbf{c} = \mathsf{GLWE}_{\sigma,n,N,Q}(\mathbf{s},\mathsf{m})$, as $\mathsf{Phase}(\mathbf{c}) = [1,-\mathbf{s}]\cdot\mathbf{c}$. We define the error of $\mathbf{c}$ as $\mathsf{Error}(\mathbf{c}) = \mathsf{Phase}(\mathbf{c}) - \mathsf{m}$.

Fully Homomorphic Encryption. Below we recall the definition of fully homomorphic encryption [60, 35].

Definition 2 (Fully Homomorphic Encryption). *A fully homomorphic encryption* FHE *consists of algorithms* (Setup, Enc, Eval, Dec) *with the following syntax.*

$\mathsf{Setup}(\lambda)$**:** *This* **PPT** *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* **PPT** *algorithm takes as input a secret key* sk*, and a message* m*, and returns a ciphertext* ct.

$\mathsf{Eval}(\mathsf{ek},[\mathsf{ct}_i]_{i=1}^n,\mathcal{C})$**:** *Given as input an evaluation key* ek*, a set of ciphertexts* $[\mathsf{ct}_i]_{i=1}^n$*, and a circuit* $\mathcal{C}$*, this (non-)deterministic algorithm outputs a ciphertext* ct.

$\mathsf{Dec}(sk,\mathsf{ct})$**:** *Given a secret key* sk *and a ciphertext* ct*, this deterministic algorithm outputs a message* m.

Correctness: *We say that* $\mathsf{FHE} = (\mathsf{Setup},\mathsf{Enc},\mathsf{Eval},\mathsf{Dec})$ *is correct, if for all security parameters* $\lambda \in \mathbb{N}$*, circuits* $\mathcal{C} : \mathcal{M}^n \mapsto \mathcal{M}$ *over the message space* $\mathcal{M}$ *of depth* $\mathsf{poly}(\lambda)$*, and 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 $\mathsf{sk} \leftarrow \mathsf{Setup}(\lambda)$*,* $\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}_i) = \mathsf{m}_i\right]_{i=1}^n$ *and* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Eval}([\mathsf{ct}_i]_{i=1}^n,\mathcal{C})$.

Efficiency: *We require that* Setup, Enc *and* Dec *run in* $\mathsf{poly}(\lambda)$ *time, and* Eval *runs in* $\mathsf{poly}(\lambda,|\mathcal{C}|)$ *time.*

Indistinguishability Under Chosen Plaintext Attack: *Let* $\lambda \in \mathbb{N}$ *be a security parameter and* $\mathsf{A} = (\mathsf{A}_0,\mathsf{A}_1)$ *be a* **PPT** *adversary. We define the advantage* $\mathsf{Adv}^{\mathsf{INDCPA}}_{\mathsf{A},\mathsf{FHE}}(\lambda)$ *We say that a* FHE *scheme is* INDCPA*-secure if for all* **PPT** *adversaries* A *the following probability*

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

is at most $\mathsf{negl}(\lambda)$*, where the oracle* $\mathcal{O}$ *on input a message* m *outputs* $\mathsf{ct} \leftarrow \mathsf{Enc}(\lambda, \mathsf{sk}, \mathsf{m})$.

Circuit Privacy: *Let* $\mathcal{C} : \mathcal{M}^n \mapsto \mathcal{M}$ *be a polynomial size circuit. A fully homomorphic encryption scheme* FHE *is said to be circuit private there exists a* **PPT** *simulator* Sim *such that*

$$\Delta(\mathsf{Sim}(\mathsf{ek}, m_{\mathsf{out}}), \mathsf{Eval}(\mathsf{ek}, c_1, \ldots, c_n, \mathcal{C})) \leq \mathsf{negl}(\lambda),$$

where $[m_i \leftarrow \mathsf{Dec}(\mathsf{sk}, c_i)]_{i=1}^n$, $m_{\mathsf{out}} \leftarrow \mathcal{C}(m_1, \ldots, m_n)$ *and* $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\lambda)$.

Our simulation-based definition of circuit privacy is stronger than [44, 13] in two aspects. First, our simulator does not require us to know the size of the circuit as in [44]. In fact, our simulator only needs to know the outcome of the circuit and nothing else. Second, the evaluator obtains as input ciphertexts that do not necessarily have to be fresh ciphertexts returned by the Enc algorithm. This way our definition captures computation on ciphertexts, that could be returned by a previous Eval invocation.

Remark 1. Note that the Definition 2 algorithm cannot satisfy circuit privacy if Eval is deterministic. We need to use the non-deterministic version of Eval to facilitate circuit privacy. Otherwise, circuit privacy can be trivially broken by running a deterministic Eval algorithm on a candidate circuit and verifying whether the outcome is the same as the outcome of the server. This attack does not require knowledge of the secret key.

3 Sanitization Bootstrapping

We describe all algorithms necessary to build the sanitization bootstrapping. For algorithms that are part of the sanitization bootstrapping but are not crucial as for the circuit privacy analysis in Section 4, we only define the interfaces and state their correctness and functionality. In Supplementary Material A we give the full specification and correctness proofs of these algorithms.

Gadgets and Gaussian Sampling. Let us first denote $\ell = \lceil \log_{\mathsf{L}} Q \rceil$ for some radix $\mathsf{L} \in \mathbb{N}$. In particular we denote L_{br} for the blind rotation key defined by Figure 2. We also use $\mathsf{L}_{\mathsf{ksK}}$ as a decomposition base of the key-switching procedure which interface we recall in Lemma 6 but device the full specification of this algorithm to Supplementary Material A.

Let $\mathbf{g}_{\mathsf{L},Q} = [1, \mathsf{L}, \ldots, \mathsf{L}^{\ell-1}]$ be the gadget vector parameterized by L and Q. We use different decomposition algorithms but refer to all with the same interface. In particular, we have the decomposition algorithm $\mathbf{x} = \mathsf{G}^{-1}_{\mathsf{ver}}(\mathfrak{c}, \mathsf{L}; \sigma) \in \mathcal{R}^\ell$ that takes as input a ring element $\mathfrak{c} \in \mathcal{R}_Q$, a radix L, and optionally a Gaussian parameter σ, and outputs a low norm vector $\mathbf{x} \in \mathcal{R}^\ell$ such that $\mathfrak{c} = \mathbf{g}^\top_{\mathsf{L},Q} \cdot \mathbf{x} \in \mathcal{R}_Q$. Note that $\mathsf{G}^{-1}_{\mathsf{ver}}$ also takes the modulus Q implicitly as input. A special case of the above is when $\mathsf{G}^{-1}_{\mathsf{ver}}$ takes as input a single element from $\mathbb{Z}_Q$ instead of a polynomial from $\mathcal{R}_Q$. We use a parameter ver that takes a value from $\{\mathsf{simul}, \mathsf{det}\}$.

If $\mathsf{ver} = \mathsf{simul}$ then we apply the algorithm from Lemma 1 coefficient wise. In particular, in our implementation we implement two algorithms from [56, 32]. One for a general $Q < \mathsf{L}^\ell$ and one specialized for $Q = \mathsf{L}^\ell$. We recall both in Supplementary Material C, but in Lemma 1 we refer only to the case with $Q < \mathsf{L}^\ell$ since, for the other case, we found it hard to find efficient parameters (despite the Gaussian sampling for $Q = \mathsf{L}^\ell$ being more efficient). We give more details on the parameters in Section 5. Note that for $\mathsf{ver} = \mathsf{simul}$, we take the additional $\sigma_{\mathbf{x}}$ as input. For $\mathsf{ver} = \mathsf{simul}$, we take the deterministic decomposition algorithm like binary decomposition but generalized to any radix $\mathsf{L} \geq 2$ and apply it coefficient-wise to elements from $\mathcal{R}_Q$. We note that for the det-mode, we may also use randomized algorithms, e.g., the subgaussian sampling algorithms [33]. We can generalize the gadget vector for some $w \in \mathbb{N}$ to a matrix $\mathbf{G}_{\mathsf{L},Q,w} = \mathbf{g}_{\mathsf{L},Q} \otimes \mathbf{I}_w \in \mathbb{Z}_Q^{w\cdot\ell\times w}$. Then the decomposition algorithm takes as input vectors $\mathbf{a} \in \mathcal{R}_Q^w$ and outputs $\mathbf{x} \in \mathcal{R}_{\mathsf{L}}^{w\cdot\ell}$ such that $\mathbf{a} = \mathbf{x}^\top \cdot \mathbf{G}_{\mathsf{L},Q,w}$.

Lemma 1 (Gaussian Sampling [32]). *There exists an algorithm* $\mathsf{G}^{-1}_{\mathsf{simul}}(a, \mathsf{L}, \sigma_{\mathbf{x}})$ *that on input* $a \in \mathbb{Z}_Q$, $\mathsf{L} \in \mathbb{N}$ *and a Gaussian parameter* $\sigma_{\mathbf{y}}$, *outputs* $\mathbf{y} \in \mathbb{Z}^\ell$ *such that*

$$\Delta\big(\mathbf{y}, \mathbf{x}[i] \in \mathcal{D}_{\Lambda^\perp_Q(\mathbf{g}_{\mathsf{L},Q})+\mathsf{G}^{-1}_{\mathsf{det}}(\mathfrak{a}[i],\mathsf{L}),\sigma_{\mathbf{x}}}\big) \geq \ell \cdot \delta,$$

if $\sigma_{\mathbf{x}} \leq \sqrt{2\mathsf{L}} \cdot (2\mathsf{L}+1) \cdot C_{\delta,\ell}$.

Depending on the ver parameter, the distribution of the image of $\mathsf{G}^{-1}_{\mathsf{ver}}$ may greatly differ. In the correctness analysis we denote the noise of $\mathsf{G}^{-1}_{\mathsf{ver}}$'s output as $\mathsf{B}(\mathsf{G}^{-1}(., \mathsf{L}))$. For example, for deterministic base-L decomposition, we take L^2, or when the decomposition returns a discrete Gaussian, we take its variance. We concretize this quantity when estimating correctness in Section 5.

Ring GSW Encryption. We recall the ring-version of the RGSW cryptosystem from Gentry, Sahai, and Waters [40] on Figure 1. We also recall the external product [21, 22], that multiplies an RGSW ciphertexts with an RLWE ciphertext. Below we state the functionality of the external product, but we limit our exposition to the case of binary plaintexts, which is the relevant case in our application.

Lemma 2 (The External Product). *Let* $\mathbf{c}$ *and* $\mathbf{C}_G$ *be as in Figure 1. If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{extProd}_{\mathsf{ver}}(\mathbf{C}_G, \mathbf{c}; \sigma_{\mathbf{x}})$ *and* $\mathfrak{m}_G \in \{0, 1\}$ *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathfrak{s}, \mathfrak{m}_{\mathsf{out}})$, *where* $\mathfrak{m}_{\mathsf{out}} = \mathfrak{m} \cdot \mathfrak{m}_G$ *and*

$$\sigma_{\mathsf{out}} \leq \sqrt{2\ell_{\mathsf{br}} \cdot N \cdot \sigma^2_{\mathsf{br}} \cdot \mathsf{B}(\mathsf{G}^{-1}_{\mathsf{ver}}(., \mathsf{L}_{\mathsf{br}}; \sigma_{\mathbf{x}})) + \mathfrak{m}_G\sigma^2}.$$

Mux Gate. Informally, the Mux gate takes as input a control RGSW sample $\mathbf{C}$ and two RLWE samples $\mathbf{d}$ and $\mathbf{h}$. The gate outputs one of the input RLWE samples depending on the bit encrypted in the RGSW sample.

Lemma 3 (Homomorphic Mux Gate). *The* Mux *algorithm takes as input* $\mathbf{C} \in \mathsf{RGSW}_{\sigma_{\mathbf{C}}}(\mathfrak{s}, m_{\mathbf{C}})$, $\mathbf{d} \in \mathsf{RLWE}_\sigma(\mathfrak{s}, \mathfrak{m}_{\mathbf{d}})$ *and* $\mathbf{h} \in \mathsf{RLWE}_\sigma(\mathfrak{s}, \mathfrak{m}_{\mathbf{h}})$, *where* $m_{\mathbf{C}} \in$

$\mathsf{RGSW}(\mathfrak{s}, \mathfrak{m}_G)$:

Input:
Secret key $\mathfrak{s} \in \mathcal{R}_Q$.
Message $\mathfrak{m}_G \in \mathcal{R}_Q$.

1 : For $i \in [\ell_{\mathsf{br}}]$:
2 : $\quad \mathbf{C}_G[*, i] \leftarrow \mathsf{RLWE}_{\sigma_G}(\mathfrak{s}, \mathfrak{m}_G \cdot \mathsf{L}_{\mathsf{br}}^{i-1})$.
3 : For $i \in [\ell_{\mathsf{br}} + 1, 2\ell_{\mathsf{br}}]$:
4 : $\quad \mathbf{C}_G[*, i] \leftarrow \mathsf{RLWE}_{\sigma_G}(\mathfrak{s}, -\mathfrak{s} \cdot \mathfrak{m}_G \cdot \mathsf{L}_{\mathsf{br}}^{i-1})$.
5 : Return $\mathbf{C}_G \in \mathcal{R}_Q^{2 \times 2\ell_{\mathsf{br}}}$.

$\mathsf{extProd}^{\mathsf{ver}}(\mathbf{c}, \mathbf{C}_G; \sigma_{\mathbf{x}})$:

Input:
Ciphertext $\mathbf{c} \in \mathsf{RLWE}_{\sigma}(\mathfrak{s}, \mathfrak{m})$.
Ciphertext $\mathbf{C}_G \in \mathsf{RGSW}_{\sigma_G}(\mathfrak{s}, \mathfrak{m}_G)$.
[If simul] A Gaussian param. $\sigma_{\mathbf{x}}$.

1 : $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{C}_G \cdot \mathsf{G}_{\mathsf{ver}}^{-1}(\mathbf{c}, \mathsf{L}_{\mathsf{br}}; \sigma_{\mathbf{x}})$.
2 : Return $\mathbf{c}_{\mathsf{out}} \in \mathcal{R}_Q^2$

Fig. 1. RGSW Encryption and External Product.

$\{0,1\}$ *and* $\mathfrak{m}_{\mathbf{d}}, \mathfrak{m}_{\mathbf{h}} \in \mathcal{R}_Q$. *Optionally it also takes a Gaussian parameter* $\sigma_{\mathbf{x}}$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{Mux}(\mathbf{C}, \mathbf{d}, \mathbf{h}; \sigma_{\mathbf{x}})$, *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathfrak{s}, \mathfrak{m}_{\mathsf{out}})$, *where* $\mathfrak{m}_{\mathsf{out}} = \mathfrak{m}_{\mathbf{d}}$ *for* $\mathsf{m}_{\mathbf{C}} = 0$ *and* $\mathsf{m}_{\mathsf{out}} = \mathfrak{m}_{\mathbf{h}}$ *for* $\mathsf{m}_{\mathbf{C}} = 1$, *and*

$$\sigma_{\mathsf{out}} \leq \sqrt{2\ell_{\mathsf{br}} \cdot N \cdot \sigma_{\mathsf{br}}^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{ver}}^{-1}(., \mathsf{L}_{\mathsf{br}}; \sigma_{\mathbf{x}})) + \sigma^2}$$

Modulus Switching and Sample Extraction. The modulus switching technique, developed by Brakerski and Vaikuntanathan [15] allows an evaluator to change the modulus of a given ciphertext without the knowledge of the secret key. Formally, we recall modulus switching by the following lemma.

Lemma 4 (Modulus Switching). *Let* $\mathbf{c} = \mathsf{LWE}_{\sigma,n,Q}(\mathbf{s}, m)$. *The modulus switching algorithm is defined as* $\mathsf{ModSwitch}(\mathbf{c}, q) = \left[\left\lfloor \frac{q \cdot \mathbf{c}[i]}{Q} \right\rceil\right]$. *If* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}, q)$, *then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\sigma_{\mathsf{out}},n,q}(\mathbf{s}, \mathsf{m} \cdot \frac{q}{Q})$, *where*

$$\sigma_{\mathsf{out}} \leq \sqrt{\left(\frac{q}{Q} \cdot \sigma\right)^2 + \frac{1}{4} \cdot \mathsf{Ha}(\mathbf{s}) \cdot \mathsf{Var}(\mathbf{s})}.$$

Furthermore, the expectation of $\mathsf{Error}(\mathbf{c}_{\mathsf{out}})$ *satisfies*

$$\left|\mathsf{E}(\mathsf{Error}(\mathbf{c}_{\mathsf{out}}))\right| \leq \left|\frac{q}{Q} \cdot \mathsf{E}(\mathsf{Error}(\mathbf{c}))\right| + \frac{1}{2}\left(1 + \mathsf{Ha}(\mathbf{s}) \cdot |\mathsf{E}(\mathbf{s})|\right)$$

Finally, if $\mathsf{m} = \mathsf{m}' \cdot \frac{Q}{t}$, *then* $\mathsf{m} \cdot \frac{q}{Q} = \mathsf{m}' \cdot \frac{q}{t}$.

Sample extraction, given by Lemma 5, allows extracting an LWE sample from an RLWE sample that encodes the constant coefficient of the RLWE sample's message.

Lemma 5 (Sample Extraction). *Let* $\mathsf{KeyExtract}(\mathfrak{s})$ *be an algorithm that on input a key* $\mathfrak{s} \in \mathcal{R}_Q$ *outputs its coefficient vector. The sample extraction algorithm*

$\mathsf{LWE\text{-}Ext}(\mathbf{c})$ *takes as input* $\mathbf{c} \in \mathsf{RLWE}_\sigma(\mathfrak{s}, \mathfrak{m})$ *and outputs* $\mathbf{c}_{\mathsf{out}} = [b, \mathbf{a}] \in \mathbb{Z}_Q^{N+1}$ *where* $b = \mathfrak{b}[1]$*, and for all* $i \in [N-1]$ *we set* $\mathbf{a}[i] \leftarrow -\mathfrak{a}[N-i+1]$ *and set* $\mathbf{a}[1] \leftarrow \mathfrak{a}[1]$.

Denote the message encoded in $\mathbf{c}$ *as* $\mathfrak{m} = \sum_{i=1}^{N} \mathfrak{m}[i] \cdot X^{i-1}$*. If* $\mathbf{s}' \leftarrow \mathsf{KeyExtract}(\mathfrak{s})$ *and* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{LWE\text{-}Ext}(\mathbf{c})$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\sigma_{\mathsf{out}},N,Q}(\mathbf{s}', \mathfrak{m}[k])$*, where* $\sigma_{\mathsf{out}} = \sigma$.

Key Switching. Key switching is an important technique for building scalable homomorphic encryption schemes. In short, by having a key switching key, the evaluator can map a given LWE sample to an LWE sample of a different key and dimension. We recall the interface for key switching and state its functionality by Lemma 6. We recall the full specification in Supplementary Material A.

Lemma 6 (Key Switching). *We define the key-switching key generation procedure* $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\sigma_{\mathsf{ksK}}, \mathbf{s}, \mathbf{s}')$*, to take as input a noise parameter* σ_{ksK}*, and LWE secret keys* $\mathbf{s} \in \mathbb{Z}_Q^n$ *and* $\mathbf{s}' \in \mathbb{Z}_Q^N$ *of (possibly) distinct dimensions* $n, N \in \mathbb{N}$*. The key-switch procedure* $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK})$ *takes as input a LWE ciphertext* $\mathbf{c} \in \mathsf{LWE}_{\sigma,N,Q}(\mathbf{s}', m)$ *and the key-switching key* ksK*, and outputs a LWE sample* $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\sigma_{\mathsf{out}},n,Q}(\mathbf{s}, m)$*, where*

$$\sigma_{\mathsf{out}} \leq \sqrt{\ell_{\mathsf{ksK}} \cdot N \cdot \mathsf{B}(\mathsf{G}_{\mathsf{det}}^{-1}(., \mathsf{L}_{\mathsf{ksK}})) \cdot \sigma_{\mathsf{ksK}}^2 + \sigma^2}.$$

Randomized Blind Rotation and Sanitization Bootstrapping. In Figure 3 we show our sanitization bootstrapping. We give the sanitizing blind rotation and its key generation algorithm in Figure 2. In short the algorithm is given a LWE sample which phase is $m + e$ (e is the noise term) and outputs a RLWE sample of $\mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e} \in \mathcal{R}_Q$. We choose the rotation polynomial $\mathfrak{a}_{\mathsf{rot}}$ such that the constant coefficient of $\mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e}$ is set to $f(m+e) \in \mathbb{Z}_Q$ for any function f that is negacyclic, i.e., satisfies $f(x + N \bmod 2N) = -f(x) \bmod Q$. We stress that the restriction on f is imposed by structural properties of the ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$. In Supplementary Material A we recall a version of the algorithm that applies a trick from [62, 52], which resolves the negacyclicity restriction on the functions that we can compute on the input plaintext at the cost of two blind rotation operations. Namely, we can program the polynomial $\mathfrak{a}_{\mathsf{rot}}$ such that $F(m+e) = \mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e}[1] \in \mathbb{Z}_Q$, where F is any function in $\mathbb{Z}_N$. Such full domain functional bootstrapping got recently much attention [50, 62, 52, 49], as it allows to compute any function on finite fields, conveniently switch from finite field plaintexts to binary and back, etc.

Lemma 7 (Correctness of Bootstrapping). *Let* br*,* $\mathbf{c}$ *and all other parameters be as in Figure 3,* ksK *be generated as described by Lemma 6, where* $\mathbf{s}' \leftarrow \mathsf{KeyExtract}(\mathfrak{s})$.

Let $\mathfrak{a}_{\mathsf{rot}}$ *be such that* $f(m+e) = \mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e}[1] \in \mathcal{R}_Q$*, where* $m + e = \mathsf{Phase}(\mathbf{c})$ *and* $f : \mathbb{Z}_{2N} \mapsto \mathbb{Z}_Q$*. If* $\mathbf{c}_{\mathsf{out}} = \mathsf{Bootstrap}^{\mathsf{ver}}(\mathsf{br}, \mathsf{ksK}, \mathbf{c}, \mathfrak{a}_{\mathsf{rot}})$*, then* $\mathbf{c}_{\mathsf{out}} \in \mathsf{LWE}_{\sigma_{\mathsf{out}},N,Q}(\mathbf{s}', f(m+e))$*, with*

$$\begin{cases} \sigma_{\mathsf{out}} \leq \sqrt{2n \cdot \ell_{\mathsf{br}} \cdot N \cdot \sigma_{\mathsf{br}}^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{det}}^{-1}(., \mathsf{L}_{\mathsf{br}}))} & \text{if } \mathsf{ver} = \mathsf{det} \\ \sigma_{\mathsf{out}} \leq \sqrt{2n \cdot \ell_{\mathsf{br}} \cdot N \cdot \sigma_{\mathsf{br}}^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{simul}}^{-1}(., \mathsf{L}_{\mathsf{br}}; \sigma_{\mathbf{x}})) + \ell_R \cdot \sigma_R^2 \cdot \sigma_{rand}^2} & \text{if } \mathsf{ver} = \mathsf{simul} \end{cases}$$

$\mathsf{BRKeyGen}(\sigma_{\mathsf{br}}, \mathfrak{s}, \mathbf{s})$:

Input:

An error distribution σ_{br}.

A RLWE secret key $\mathfrak{s} \in \mathcal{R}_Q$.

A LWE secret key $\mathbf{s} \in \mathbb{Z}_t^n$.

1 : For $i \in [n]$

2 : Set $\mathsf{br}[i] = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, \mathbf{s}[i])$.

3 : Output $\mathsf{br} \in \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, .)^n$.

$\mathsf{BlindRotate}^{\mathsf{ver}}(\mathsf{br}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{c}; \sigma_{\mathbf{x}})$:

Input:

A blind rotation key $\mathsf{br} = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, .)^n$.

An rotation polynomial $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$.

A ciphertext $\mathbf{c} \in \mathsf{LWE}_{\sigma,n,2N}(\mathbf{s}, .)$.

[If simul] A Gaussian param. $\sigma_{\mathbf{x}}$.

1 : Let $\mathbf{c} = [b, \mathbf{a}] \in \mathbb{Z}_{2N}^{n+1}$.

2 : Set $\mathbf{c}_{\mathsf{acc},0} \leftarrow [\mathfrak{a}_{\mathsf{rot}} \cdot X^b, 0] \in R_Q^2$

3 : For $i \in [n]$:

4 : $\mathbf{c}_{\mathsf{acc},i} \leftarrow \mathsf{Mux}^{\mathsf{ver}}(\mathsf{br}[i], \mathbf{c}_{\mathsf{acc},i-1} \cdot X^{-\mathbf{a}[i]}, \mathbf{c}_{\mathsf{acc},i-1}; \sigma_{\mathbf{x}})$.

5 : Output $\mathbf{c}_{\mathsf{acc},n} \in R_Q^2$.

Fig. 2. TFHE-style Blind Rotation and its Setup.

Additionally, we have that $\mathbf{c}_{\mathsf{in}}$ *from step 2 on Figure 3 is such that* $\mathbf{c}_{\mathsf{in}} = \mathsf{LWE}_{\sigma_{\mathsf{in}},n,2N}(\mathbf{s}, .)$*, where*

$$\sigma_{\mathsf{in}} \leq \sqrt{\left(\frac{q}{Q} \cdot \sigma_1\right)^2 + \frac{1}{4} \cdot \mathsf{Ha}(\mathbf{s}) \cdot \mathsf{Var}(\mathbf{s})}$$

with $\sigma_1 \leq \sqrt{N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{B}(\mathsf{G}_{\mathsf{ver}}^{-1}(., \mathsf{L}_{\mathsf{ksK}})) \cdot \sigma_{\mathsf{ksK}}^2 + \sigma_{\mathsf{out}}^2}$.

The full cryptosystem. Below we briefly describe how the complete cryptosystem fits into Definition 2.

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 $\mathbf{s}' \leftarrow \mathsf{KeyExtract}(\mathfrak{s})$, and $\mathbf{s} \in \{0,1\}^n$ for the LWE key[2]. Choose the radices L_{br}, $\mathsf{L}_{\mathsf{ksK}}$, $\mathsf{L}_R \in \mathbb{N}$ and the Gaussian parameters σ, σ_{ksK}, σ_{br}, σ_R, σ_{rand}, $\sigma_{\mathbf{x}} > 0$. Run $\mathsf{br} \leftarrow \mathsf{BRKeyGen}(\sigma_{\mathsf{br}}, \mathfrak{s}, \mathbf{s})$, $\mathsf{ksK} \leftarrow \mathsf{KeySwitchSetup}(\sigma_{\mathsf{ksK}}, \mathbf{s}, \mathbf{s}')$, and $\mathbf{v} \leftarrow \mathsf{LWE}_{\sigma_R,n,Q}(\mathbf{s}', 0)^{\ell_R}$, where $\ell_R = \log_{\mathsf{L}_R}(Q)$. Finally, set the evaluation key $\mathsf{ek} = (\mathsf{br}, \mathsf{ksK}, \mathbf{v})$ and the secret key $\mathsf{sk} = (\mathfrak{s}, \mathbf{s}', \mathbf{s})$.

Encryption: To encrypt a message $m' \in \mathbb{Z}_t$ we compute $\mathbf{c} = \mathsf{LWE}_{\sigma,n,q}(\mathbf{s}', m) \in \mathbb{Z}_Q^{N+1}$, where $m = \frac{Q}{t} \cdot m' \in \mathbb{Z}_Q$. Note that we can also use the vector $\mathbf{v}$ to obtain a LWE sample that is close to a "fresh" one, and then we simply add m. Finally, we note that the ciphertexts that are input to the

[2] Other distributions for the LWE secret key are possible. See [57] for an excelent summary.

$\mathsf{Bootstrap}^{\mathsf{ver}}(\mathsf{br}, \mathsf{ksK}, \mathbf{c}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{v}, \sigma_{\mathbf{rand}}, \sigma_{\mathbf{x}})$:

Input:

A blind rotation key $\mathsf{br} = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, .)^n$.

A key switch key $\mathsf{ksK} \in \mathsf{LWE}_{\sigma_{\mathsf{ksK}},n,Q}(\mathbf{s}, .)^{\ell_{\mathsf{ksK}} N}$.

Ciphertext $\mathbf{c} = \mathsf{LWE}_{\sigma,n,q}(\mathbf{s}', .) \in \mathbb{Z}_Q^{N+1}$, where $\mathbf{s}' = \mathsf{KeyExtract}(\mathfrak{s})$.

A rotation polynomial $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$.

A vector $\mathbf{v} = \mathsf{LWE}_{\sigma_R,n,Q}(\mathbf{s}', 0)^{\ell_R}$, where $\ell_R = \log_{\mathsf{L}_R}(Q)$.

[If simul] Gaussian parameters $\sigma_{\mathbf{rand}}, \sigma_{\mathbf{x}}$.

1 : Run $\mathbf{c}_{\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK}) \in \mathbb{Z}_Q^{n+1}$.

2 : Run $\mathbf{c}_{\mathsf{in}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{\mathsf{ksK}}, 2N) \in \mathbb{Z}_{2N}^{n+1}$.

3 : Run $\mathbf{c}_{\mathsf{acc}} \leftarrow \mathsf{BlindRotate}^{\mathsf{ver}}(\mathsf{br}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{c}_{\mathsf{in}}, \sigma_{\mathbf{x}})$.

4 : Run $\mathbf{c}_{\mathsf{ext}} \leftarrow \mathsf{LWE\text{-}Ext}(\mathbf{c}_{\mathsf{acc}})$.

5 : If $\mathsf{ver} = \mathsf{simul}$:

6 : Choose $\mathbf{r} \leftarrow_\$ \mathcal{D}_{\Lambda_Q^\perp(\mathbf{g}_{\mathsf{L}_R,Q})+0,\sigma_{\mathbf{rand}}}$, $r \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{rand}}}$ and $y \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}}$.

7 : Set $\mathbf{c}_{\mathbf{rand}} \leftarrow \mathbf{r}^\top \cdot \mathbf{v} + r$.

8 : Set $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c}_{\mathsf{ext}} + \mathbf{c}_{\mathbf{rand}} + y$.

9 : Otherwise set $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c}_{\mathsf{ext}}$.

10 : Return $\mathbf{c}_{\mathsf{out}} \in \mathbb{Z}_Q^{N+1}$.

Fig. 3. Bootstrapping.

circuit could be generated with the lower dimension n and modulus $2N$. We bootstrapping these ciphertexts, we can omit the key and modulus switching steps at Figure 3.

Eval: We can represent homomorphic computation as a circuit with gates of the form $f(b + \sum_{i=1}^{k} x_i \cdot a_i \in \mathbb{Z}_{t_1}) \in \mathbb{Z}_{t_2}$ 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 $f : \mathbb{Z}_{t_1} \mapsto \mathbb{Z}_{t_2}$ by applying the bootstrapping algorithm from Figure 3. The class of functions f that we can compute is restricted to negacyclic functions, but we can extend the bootstrapping procedure to support arbitrary functions (see Supplementary Material A). We compute all gates with $\mathsf{ver} = \mathsf{det}$ except for the output gates, where we run the sanitization bootstrap with $\mathsf{ver} = \mathsf{simul}$. Crucially, the evaluator should finish the computation with a sanitization bootstrap to achieve circuit privacy. We can continue computing on the sanitized ciphertexts, only if the following computation is public and known to the client. Otherwise, the evaluator should sanitize the ciphertexts again.

Decryption: Do decrypt a 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 $\left\lceil \frac{t}{Q}(\frac{Q}{t}m'_{\mathsf{out}} + e) \right\rfloor = m'_{\mathsf{out}}$ if $|e| \leq \frac{Q}{2t}$.

4 Analysis of Circuit Privacy

This section contains our core analytical contribution. First, we recall some additional mathematical preliminaries.

Definition 3 (Smoothing Parameter). *For a lattice $\Lambda \subseteq \mathbb{Z}^m$ and positive real $\delta > 0$, the smoothing parameter η_δ is the smallest real $r > 0$ such that $\rho_{1/r}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \delta$, where $\Lambda^* = \{\mathbf{x} \in \mathbb{R}^m | \mathbf{x}^{\top}\Lambda \subseteq \mathbb{Z}\}$.*

Lemma 8 ([55], Lemma 3.3). *Let Λ be any rank-m lattice, and $\delta \in \mathbb{R}^+$. Then*

$$\eta_\delta \leq \lambda_m(\Lambda) \cdot C_{\delta,m},$$

where $\lambda_m(\Lambda)$ is the smallest R such that the ball $\mathcal{B}_R$ centered in the origin and with radius R contains m linearly independent vectors of Λ. Remind that we denote $C_{\delta,m} = \sqrt{\frac{\ln(2m(1+1/\delta))}{\pi}}$.

4.1 Generalized Gaussian Leftover Hash and Core Randomization Lemma.

Below we give our fixed and generalized version of the Gaussian Leftover hash lemma from [13].

Lemma 9 (Gaussian Leftover Hash Lemma (Generalized Lemma 3.6 from [13])). *Let δ, $\sigma_{\mathbf{x}} > 0$. Let $\mathcal{L} = \{\mathbf{v} = \hat{\Lambda} : \hat{\mathbf{e}}^{\top} \cdot \mathbf{v} = 0\}$, where $\hat{\mathbf{e}} = [\mathbf{e}, 1] \in \mathbb{Z}^{m+1}$, $\hat{\Lambda} = \hat{\Lambda}_Q^{\perp}(\mathbf{G}_{\mathsf{L},Q,w}) \times \mathbb{Z}$, $m = w \cdot \ell$ and $\mathsf{L}^{\ell} \leq Q$. Let $\mathbf{q}_{\mathsf{L},Q} = [q_i]_{i=1}^{\ell}$ be the base-L decomposition of Q for $Q < \mathsf{L}^{\ell}$, and $\mathbf{q}_{\mathsf{L},Q} = [\mathbf{0}, \mathsf{L}]$ for $Q = \mathsf{L}^{\ell}$. For any $\mathbf{e} \in \mathbb{Z}^m$ and $\mathbf{c} \in \mathbb{R}^m$, if*

$$\sigma_{\mathbf{x}} \geq \sqrt{1 + ||\hat{\mathbf{e}}||_\infty} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}^2 + 1}\right) \cdot C_{\delta,m}, \text{ then}$$

$$\Delta(\mathbf{e}^{\top}\mathbf{x} + y, e') < 2\delta,$$

where $\mathbf{x} \leftarrow_{\$} \mathcal{D}_{\Lambda_Q^{\perp}(\mathbf{G}_{\mathsf{L},Q,w})+\mathbf{c},\sigma_{\mathbf{x}}}$, $y \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}}$, $e' \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}} \cdot \sqrt{1+||\mathbf{e}||^2}}$, and $m = w \cdot \ell$ with $\ell \in \mathbb{N}$. Note that the distribution of e' is independent of the coset $\mathbf{c}$, and if $\mathbf{e} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{br}}}$, then $e' \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}} \cdot \sqrt{1+m\sigma_{\mathsf{br}}^2}}$.

The proof of the lemma follows from a technical lemma (Corollary 2.8 in [39]), and a lemma (Lemma 3.7 in [13]) that bounds the smoothing parameter η_δ for the lattice $\mathcal{L} = \{\mathbf{v} = \hat{\Lambda} : \hat{\mathbf{e}}^{\top} \cdot \mathbf{v} = 0\}$, where $\hat{\mathbf{e}} = [\mathbf{e}, 1] \in \mathbb{Z}^{m+1}$. In [13], they prove the lemma for a modulus Q that is a power of two. We generalize the lemma to moduli of form $Q \leq \mathsf{L}^{\ell}$.

Lemma 10 (Generalized Lemma 3.7 from [13]). *Let $\delta > 0$. Let $\mathcal{L} = \{\mathbf{v} = \hat{\Lambda} : \hat{\mathbf{e}}^\top \cdot \mathbf{v} = 0\}$, where $\hat{\mathbf{e}} = [\mathbf{e}, 1] \in \mathbb{Z}^{m+1}$, $\hat{\Lambda} = \hat{\Lambda}_Q^\perp(\mathbf{G}_{\mathsf{L},Q,w}) \times \mathbb{Z}$, $m = w \cdot \ell$ and $\mathsf{L}^\ell \leq Q$. Furthermore, let $\mathbf{q}_{\mathsf{L},Q} = [q_i]_{i=1}^\ell$ be the base-L decomposition of Q for $Q < \mathsf{L}^\ell$, and $\mathbf{q}_{\mathsf{L},Q} = [\mathbf{0}, \mathsf{L}]$ for $Q = \mathsf{L}^\ell$. Then we have*

$$\eta_\delta \leq \sqrt{1 + ||\hat{\mathbf{e}}||_\infty} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}^2 + 1}\right) \cdot C_{\delta,m}.$$

Proof. We use Lemma 8 to bound the smoothing parameter of $\mathcal{L}$. Since $\hat{\Lambda} = \hat{\Lambda}_Q^\perp(\mathbf{G}_{\mathsf{L},Q,w}^\top) \times \mathbb{Z}$ is of dimension $m+1$ and $\mathcal{L}$ is a sub-lattice of $\hat{\Lambda}$ made of the vectors that are orthogonal to $\mathbf{e}$, we have that $\mathcal{L}$ is of dimension m. We thus exhibit m independent short vectors of $\mathcal{L}$ to obtain an upper bound on $\lambda_m(\mathcal{L})$. We first define the matrix

$$\bar{\mathbf{B}} = \begin{bmatrix} \mathsf{L} & & & q_1 \\ -1 & \mathsf{L} & & q_2 \\ & -1 & \ddots & \vdots \\ & & \ddots & \mathsf{L} & q_{\ell-1} \\ & & & -1 & q_\ell \end{bmatrix} \in \mathbb{Z}^{\ell \times \ell},$$

where $\mathbf{q}_{\mathsf{L},Q} = [q_i]_{i=1}^\ell$ is the base-L decomposition of the modulus Q if $Q < \mathsf{L}^\ell$, and $q_\ell = \mathsf{L}$ and $q_i = 0$ for $i < \ell$ if $Q = \mathsf{L}^\ell$. Note that $\bar{\mathbf{B}}$ is a basis for the lattice $\Lambda_Q^\perp(\mathbf{g}_{\mathsf{L},Q})$. The lattice $\hat{\Lambda}$ is then generated by the columns of the matrix:

$$\mathbf{B} = [\mathbf{b}_1 | \dots | \mathbf{b}_{m+1}] = \begin{bmatrix} \mathbf{I}_w \otimes \bar{\mathbf{B}} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{Z}^{(m+1)\times(m+1)}$$

For $k \leq m$ let $\mathbf{u}_k = \mathbf{b}_k - \mathbf{b}_{m+1} \cdot \hat{\mathbf{e}}^\top \cdot \mathbf{b}_k$. Since $\hat{\mathbf{e}}^\top \cdot \mathbf{b}_{m+1} = 1$ we directly have $\mathbf{e}^\top \cdot \mathbf{u}_k = 0$ and thus $\mathbf{u}_k \in \mathcal{L}$. The vectors $\mathbf{u}_1, \dots, \mathbf{u}_m$ are linearly independent since $\mathsf{span}(\mathbf{u}_1, \dots, \mathbf{u}_m, \mathbf{b}_{m+1}) = \mathsf{span}(\mathbf{b}_1, \dots, \mathbf{b}_m, \mathbf{b}_{m+1}) = \mathbb{R}^{m+1}$ (which comes from the fact that $\mathbf{B}$ is a basis of an $(m+1)$-dimensional lattice).

We now bound the norm of $\mathbf{u}_k$. Note that $\mathbf{b}_{m+1} \cdot \hat{\mathbf{e}}^\top \leq ||\hat{\mathbf{e}}||_\infty$, because $\mathbf{b}_{m+1} = [\mathbf{0}, 1]$. Then we have

$$\begin{aligned} ||\mathbf{u}_k|| &= ||\mathbf{b}_k - \mathbf{b}_{m+1} \cdot \hat{\mathbf{e}}^\top \cdot \mathbf{b}_k|| \\ &\leq ||\mathbf{b}_k + ||\hat{\mathbf{e}}||_\infty \cdot \mathbf{b}_k|| \\ &= ||(1 + ||\hat{\mathbf{e}}||_\infty)\mathbf{b}_k|| \\ &= \sqrt{1 + ||\hat{\mathbf{e}}||_\infty} \cdot ||\mathbf{b}_k||. \end{aligned}$$

What is left to do is to bound the norm of $\mathbf{b}_k$. Note that for $k < m+1$ the vector $\mathbf{b}_k$ has L in its kth position, -1 in position $k+1$, and 0 in all other positions. Furthermore, the vectors $\mathbf{b}_k$ for $k = 0 \mod \log_{\mathsf{L}}(Q)$ contain the vector $\mathbf{q}_{\mathsf{L},Q}$ and are zero at all other positions. Hence, we can bound the norm by $||\mathbf{b}_k|| \leq \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}^2 + 1}\right)$. In particular, for $\mathsf{L}^{\ell+1} = Q$ the norm of $\mathbf{b}_k$ is bounded by $\sqrt{\mathsf{L}^2 + 1}$, while for $\mathsf{L}^{\ell+1} > Q$ the bound depends on the decomposition of Q.

To summarize we obtain

$$\lambda_m(\mathcal{L}) \leq \max_{k \leq m} ||\mathbf{u}_k|| \leq \sqrt{1 + ||\hat{\mathbf{e}}||_\infty} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}^2 + 1}\right).$$

Below we recall the Core Randomization Lemma from [13], but we update it with our Lemma 9.

Lemma 11 (Core Randomization Lemma (Updated Lemma 3.1 from [13])). *Let $\gamma, \epsilon > 0$. Let $\mathbf{q}_{\mathsf{L},Q} = [q_i]_{i=1}^{\ell}$ be the base-L decomposition of Q for $Q < \mathsf{L}^{\ell}$, and $\mathbf{q}_{\mathsf{L},Q} = [\mathbf{0}, \mathsf{L}]$ for $Q = \mathsf{L}^{\ell}$. For any $\mathbf{e} \in \mathbb{Z}_Q^m$ and $\mathbf{c} \in \mathbb{R}^m$, if*

$$\sigma_{\mathit{rand}} \geq \max\left(4\big((1-\gamma)(2\epsilon)^2\big)^{-1/m}, \sqrt{1+||\hat{\mathbf{e}}||_\infty} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}^2+1}\right) \cdot C_{\gamma,m}\right)$$

then

$$\Delta\big((\mathbf{A}, \mathbf{Ar}, \mathbf{e}^\top\mathbf{r} + r), (\mathbf{A}, \mathbf{u}, e')\big) \leq \epsilon + 2\gamma$$

where $\mathbf{r} \leftarrow_\$ \mathcal{D}_{\Lambda_Q^\perp(\mathbf{G}_{\mathsf{L},Q,w})+\mathbf{c},\sigma_{\mathit{rand}}}$, $\mathbf{A} \leftarrow_\$ \mathbb{Z}_Q^{(w-1)\times m}$, $\mathbf{u} \leftarrow_\$ \mathbb{Z}_Q^{w-1}$, $r \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathit{rand}}}$, $e' \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathit{rand}}\sqrt{1+||\mathbf{e}||^2)}}$ and $m = w \cdot \ell$. Furthermore, if $\mathbf{e} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_R}$, then $e' \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathit{rand}}\cdot\sqrt{1+m\sigma_R^2}}$.

4.2 Distribution of Our Randomized Bootstrapping and Circuit Privacy

Below we state and prove the core theorem on the distribution of bootstrapped ciphertexts. Circuit privacy, that we prove at the end of this section, follows nearly immediately from the theorem below.

Theorem 1 (Distribution of the Bootstrap). *Let br be the blind rotation key, $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$ a rotation polynomial, and $\mathbf{c} \in \mathsf{LWE}_\sigma(\mathbf{s}, m)$ a LWE sample as defined in the* Bootstrap *algorithm in Figure 2. Assume that $\mathfrak{a}_{\mathsf{rot}}$ is such that $f(m) = (\mathfrak{a}_{\mathsf{rot}} \cdot X^{\mathsf{Phase}(\mathbf{c}_{\mathsf{in}})})[1]$ where $\mathbf{c}_{\mathsf{in}}$ is the LWE sample obtained at step 2 of the* Bootstrap *algorithm. Let $\mathbf{c}_{\mathsf{out}}$ be the LWE sample returned by the* Bootstrap *algorithm for $\mathsf{ver} = \mathsf{simul}$ and Gaussian parameters σ_{rand} and $\sigma_{\mathbf{x}}$ where the Gaussian sampling algorithm $\mathsf{G}_{\mathsf{simul}}^{-1}$ is as in Lemma 1. Assume that*

$$\sigma_{\mathit{rand}} \geq \max\left(4\big((1-\gamma)(2\epsilon)^2\big)^{-1/\ell_R}, \sqrt{1+B_R} \cdot \max\left(||\mathbf{q}_{\mathsf{L}_R,Q}||, \sqrt{\mathsf{L}_R^2+1}\right) \cdot C_{\gamma,\ell_R}\right)$$

and

$$\sigma_{\mathbf{x}} \geq \sqrt{1+B_{\mathsf{br}}} \cdot \max\left(||\mathbf{q}_{\mathsf{L}_{\mathsf{br}},Q}||, \sqrt{\mathsf{L}_{\mathsf{br}}^2+1}\right) \cdot C_{\delta,2\cdot n\cdot N\cdot \ell_{\mathsf{br}}},$$

where B_{br} and B_R are bounds on the infinity norm of the noise terms in the blind rotation key br and the masking vector $\mathbf{v}$. Then we have

$$\Delta(\mathbf{c}_{\mathsf{out}}, \mathbf{c}_{\mathsf{fresh}}) \leq \max\big(\epsilon + 2\gamma, 2\delta\big),$$

where $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$, $b_{\mathsf{fresh}} = \langle \mathbf{a}_{\mathsf{fresh}}, \mathbf{s}' \rangle + f(m) + e_{\mathit{rand}} + e_{\mathsf{out}}$, $e_{\mathit{rand}} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathit{rand}}\cdot\sqrt{1+\ell_R\sigma_R^2}}$, and $e_{\mathsf{out}} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}\sqrt{1+2nN\sigma_{\mathsf{br}}}}$.

Proof. The proof consists of two parts. First we analyze the LWE sample that is extracted after blind rotation. In particular, we give a concise representation of the final noise term. Furthermore, we show that each noise coefficient and decomposition term of the randomized decomposition appears only once in the final noise term. The second part of the proof consists of a hybrid argument, where we argue step-by-step that the distribution of the extracted and "masked" LWE sample is statistically close to a "freshly" sampled LWE sample of the same message.

Below we give the first part of the proof. But to further tame complexity we split this part in three more sub-parts. First we analyze a single external product, then a single MUX gate and we finalize this part with blind rotation and extraction.

Single External Product. First let us remind that for $j \in [\ell_{\mathsf{br}}]$ we have

$$\begin{aligned}\mathbf{C}_G[*, j] &= \mathsf{RLWE}_\sigma(\mathfrak{s}, \mathfrak{m}_G \cdot \mathsf{L}_{\mathsf{br}}^{j-1}) \text{ and}\\ \mathbf{C}_G[*, j+\ell_{\mathsf{br}}] &= \mathsf{RLWE}_\sigma(\mathfrak{s}, -\mathfrak{s} \cdot \mathfrak{m}_G \cdot \mathsf{L}_{\mathsf{br}}^{j-1}).\end{aligned}$$

Denote $\mathbf{C}_G = [\mathfrak{a}_j, \mathfrak{b}_j]_{j=1}^{2\ell_{\mathsf{br}}}$, and $\mathbf{d} = [\mathfrak{a}_d, \mathfrak{b}_d]$ where $\mathfrak{b}_d = \mathfrak{a}_d \cdot \mathfrak{s} + \mathfrak{e}_d + \mathfrak{m}_d$. We analyze the sample $\mathbf{c}_{\mathsf{prod}} \leftarrow \mathsf{extProd}^{\mathsf{simul}}(\mathbf{C}_G, \mathbf{d})$. Then in step 1 and 2 of the $\mathsf{extProd}^{\mathsf{simul}}$ algorithm we compute $\mathbf{c}_{\mathsf{prod}} = \mathbf{C}_G \cdot \mathsf{G}_{\mathsf{simul}}^{-1}(\mathbf{d}, \mathsf{L}_{\mathsf{br}}) = [\mathfrak{a}_{\mathsf{prod}}, \mathfrak{b}_{\mathsf{prod}}]$. Let us denote the vector $[\mathfrak{x}_j]_{j=1}^{2\ell_{\mathsf{br}}} = \mathsf{G}_{\mathsf{simul}}^{-1}(\mathbf{d}, \mathsf{L}_{\mathsf{br}})$. We can write

$$\mathfrak{a}_{\mathsf{prod}} = \sum_{j=1}^{2\ell_{\mathsf{br}}} \mathfrak{a}_j \cdot \mathfrak{x}_j.$$

Furthermore, we can write

$$\mathfrak{b}_{\mathsf{prod}} = \mathfrak{a}_{\mathsf{prod}} \cdot \mathfrak{s} + \mathfrak{m}_G \cdot \mathfrak{e}_d + \widehat{\mathfrak{e}} + \mathfrak{m}_G \cdot \mathfrak{m}_g, \tag{1}$$

where $\widehat{\mathfrak{e}} = \sum_{j=1}^{2\ell_{\mathsf{br}}} \mathfrak{e}_j \cdot \mathfrak{x}_j$. Equation 1 holds because we have

$$\begin{aligned}\sum_{j=1}^{2\ell_{\mathsf{br}}} \mathfrak{b}_j \cdot \mathfrak{x}_j &= \mathfrak{a}_{\mathsf{out}} \cdot \mathfrak{s} + \sum_{j=1}^{\ell_{\mathsf{br}}} (\mathfrak{e}_j + \mathfrak{m}_G \cdot \mathsf{L}^{j-1}) \cdot \mathfrak{x}_j + \sum_{j=1}^{\ell_{\mathsf{br}}} (\mathfrak{e}_{j+\ell_{\mathsf{br}}} - \mathfrak{s} \cdot \mathfrak{m}_G \cdot \mathsf{L}^{j-1}) \cdot \mathfrak{x}_{j+\ell_{\mathsf{br}}}\\ &= \mathfrak{a}_{\mathsf{out}} \cdot \mathfrak{s} + \sum_{j=1}^{2\ell_{\mathsf{br}}} \mathfrak{x}_j \cdot \mathfrak{e}_j + \mathfrak{m}_G \cdot \Big(\sum_{j=1}^{\ell_{\mathsf{br}}} \mathfrak{x}_j \cdot \mathsf{L}^{j-1} - \mathfrak{s} \cdot \sum_{j=\ell_{\mathsf{br}+1}}^{2\ell_{\mathsf{br}}} \mathfrak{x}_j \cdot \mathsf{L}^{j-1} \Big)\end{aligned}$$

and in particular from the properties of the Gaussian sampling algorithm (Lemma 1) we have that

$$\mathfrak{m}_G \cdot \Big(\sum_{j=1}^{\ell_{\mathsf{br}}} \mathfrak{x}_j \cdot \mathsf{L}^{j-1} - \mathfrak{s} \cdot \sum_{j=\ell_{\mathsf{br}+1}}^{2\ell_{\mathsf{br}}} \mathfrak{x}_j \cdot \mathsf{L}^{j-1} \Big) = \mathfrak{m}_G(\mathfrak{b}_d - \mathfrak{a}_d \cdot \mathfrak{s}) = \mathfrak{m}_G(\mathfrak{e}_d + \mathfrak{m}_d).$$

Single MUX Gate. Let us analyze a single execution of the MUX gate $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{Mux}(\mathbf{C}_G, \mathbf{d}, \mathbf{h})$, where $\mathbf{C}_G$ and $\mathbf{d}$ are RGSW and RLWE samples as above, and

$\mathbf{h} \in \mathsf{RLWE}_\sigma(\mathfrak{s}, \mathfrak{m}_\mathbf{h}) = [\mathfrak{a}_h, \mathfrak{b}_h]$. Since $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{extProd}^{\mathsf{simul}}(\mathbf{C}, \mathbf{d}) + \mathbf{h}$, we can write $\mathbf{c}_{\mathsf{out}} = [\mathfrak{a}_{\mathsf{prod}}, \mathfrak{b}_{\mathsf{prod}}] + [\mathfrak{a}_h, \mathfrak{b}_h]$. Remind that the message encoded in $\mathbf{C}_G$ is a single bot $\mathfrak{m}_G \in \{0, 1\}$. That is, we are only interested in the special case where the RGSW message is a single bit. We have two cases:

- The case with $\mathfrak{m}_G = 0$. In this case we can write $\mathfrak{b}_{\mathsf{prod}} = \mathfrak{a}_{\mathsf{prod}} \cdot \mathfrak{s} + \mathfrak{e}$. In other words, the error from $\mathbf{d}$ cancels out. And we have that $\mathfrak{b}_{\mathsf{out}} = \mathfrak{b}_{\mathsf{prod}} + \mathfrak{b}_h = \mathfrak{a}_{\mathsf{out}} \cdot \mathfrak{s} + \mathfrak{e} + \mathfrak{m}_h + \mathfrak{e}_h$.
- The case with $\mathfrak{m}_G = 1$. In this case we can write $\mathfrak{b}_{\mathsf{prod}} = \mathfrak{a}_{\mathsf{prod}} \cdot \mathfrak{s} + \mathfrak{e} + \mathfrak{m}_d - \mathfrak{m}_h + \mathfrak{e}_d - \mathfrak{e}_h$. And we have that $\mathfrak{b}_{\mathsf{out}} = \mathfrak{b}_{\mathsf{prod}} + \mathfrak{b}_h = \mathfrak{a}_{\mathsf{out}} \cdot \mathfrak{s} + \mathfrak{m}_d + \mathfrak{e} + \mathfrak{e}_d$.

Finally, note that in blind rotation we have $\mathbf{d} = \mathbf{h} \cdot X^l \in \mathcal{R}_Q^2$ for some $l \in \mathbb{Z}_{2N}$. That is, the two ring LWE samples are negacyclic rotations of one another. This means that $\mathfrak{e}_d = \mathfrak{e} \cdot X^l \in \mathcal{R}_Q$.

Blind Rotation and Extraction. First we set the accumulator to $\mathbf{c}_{\mathsf{acc},0} = [0, \mathfrak{a}_{\mathsf{rot}} \cdot X^b]$. Note that the first accumulator is special because its noise term is zero. Let us denote the error term that is added in the ith iteration of the blind rotation loop by $\widehat{\mathfrak{e}}_i$. Particularly, this noise term is $\widehat{\mathfrak{e}}_i = \sum_{j=1}^{2\ell_{\mathsf{br}}} \mathfrak{e}_{i,j} \cdot \mathfrak{x}_{i,j}$, where $\mathfrak{e}_{i,j}$ is the noise term of the blind rotation keys, $\mathfrak{x}_{i,j}$ is the component from the randomized decomposition algorithm. At the ith iteration we run a homomorphic MUX gate that multiplies the input RLWE sample's noise and message by $X^{-\mathbf{a}[i] \cdot \mathbf{s}[i]}$, and adds a new noise term $\widehat{\mathfrak{e}}_i$. Remind that the noise term of $\mathbf{c}_{\mathsf{acc},0}$ is zero, thus at iteration $i = 1$, the resulting RLWE sample $\mathbf{c}_{\mathsf{acc},1}$ has noise term $\widehat{\mathfrak{e}}_1$. Then $\mathbf{c}_{\mathsf{acc},1}$ has noise term $\widehat{\mathfrak{e}}_1 \cdot X^{-\mathbf{a}[2] \cdot \mathbf{s}[2]} + \widehat{\mathfrak{e}}_2$. Finally after n iterations we have

$$\mathsf{Error}(\mathbf{c}_{\mathsf{acc}}) = \mathsf{Error}(\mathbf{c}_{\mathsf{acc},n}) = \sum_{i=1}^{n} \widehat{\mathfrak{e}}_i \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]} \tag{2}$$

and the message is $\mathfrak{a}_{\mathsf{rot}} \cdot X^{\mathsf{Phase}(\mathbf{c})}$.

Let us denote $\mathbf{c}_{\mathsf{acc}} = [\mathfrak{a}_{\mathsf{acc}}, \mathfrak{b}_{\mathsf{acc}}]$ and the extracted LWE sample $\mathbf{c}_{\mathsf{ext}} = [\mathbf{a}_{\mathsf{ext}}, b_{\mathsf{ext}}]$. Note that $\mathbf{a}_{\mathsf{ext}} \in \mathbb{Z}_Q^N$ and $b_{\mathsf{ext}} = \mathfrak{b}_{\mathsf{acc}}[1] = \langle \mathbf{a}_{\mathsf{acc}}, \mathbf{s}' \rangle + \mathfrak{m} \cdot X^{\mathsf{Phase}(\mathbf{c})}[1] + (\sum_{i=1}^{n} \widehat{\mathfrak{e}}_i \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]})[1]$. In particular, note that

$$\begin{aligned}
\mathsf{Error}(\mathbf{c}_{\mathsf{ext}}) &= \Big(\sum_{i=1}^{n} \widehat{\mathfrak{e}}_i \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]} \Big)[1] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{2\ell_{\mathsf{br}}} \big(\mathfrak{e}_{i,j} \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]} \cdot \mathfrak{x}_{i,j} \big)[1] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{2\ell_{\mathsf{br}}} \sum_{k=1}^{N} \mathbf{e}_{i,j}[k] \cdot \mathfrak{x}_{i,j}[k]
\end{aligned}$$

where $\mathbf{e}_{i,j}$ is a vector of discrete Gaussian random variables centered at zero of parameter σ_{br}. Note that in the ring $\mathcal{R}_Q$ and assuming the coefficients of $\mathfrak{e}_{i,j}$ are centered at zero, we have that $\mathfrak{e}_{i,j} \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]} = \mathfrak{e}'_{i,j}$ rotates the coefficients of $\mathfrak{e}_{i,j}$ negacyclicly, meaning that $\mathfrak{e}'_{i,j}$ has the same distribution as

$\mathfrak{e}_{i,j}$. Similarly, we have that the constant coefficient of $\mathfrak{e}'_{i,j} \cdot \mathfrak{x}_{i,j}$ in $\mathcal{R}_Q$ is $(\mathfrak{e}'_{i,j} \cdot \mathfrak{x}_{i,j})[1] = \mathfrak{x}_{i,j}[1] \cdot \mathfrak{e}'_{i,j}[1] + \sum_{k=2}^{N} -\mathfrak{e}'_{i,j}[N-k+1] \cdot \mathfrak{x}_{i,j}[k]$. Hence we can write $\mathbf{e}_{i,j}[k] = -\mathfrak{e}'_{i,j}[N-k+1]$ for $k \in [2, N]$ and $\mathbf{e}_{i,j}[1] = \mathfrak{e}'_{i,j}[1]$. Therefore $\mathbf{e}_{i,j}[k]$ is is from the discrete Gaussian distribution of the same parameter as $\mathfrak{e}_{i,j}$ given that the distribution of the coefficients are centered at zero.

Distribution of the Extracted LWE Sample. Now we are ready to argue that the $\mathbf{c}_{\mathsf{out}} = \mathbf{c}_{\mathsf{ext}} + \mathbf{r}^\top \cdot \mathbf{v} + r + y$ sample is statistically close to a LWE sample of the same message that is independent of the input ciphertext. The proof is due to the following hybrid argument.

Hybrid 0. In this hybrid the sample is as in the original scheme. Specifically, we have $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c}_{\mathsf{ext}} + \mathbf{c}_{\mathsf{rand}} + y$, where $\mathbf{c}_{\mathsf{rand}} \leftarrow \mathbf{r}^\top \cdot \mathbf{v} + r$ with $\mathbf{r} \leftarrow_{\$} \mathcal{D}_{\Lambda^\perp_Q(\mathbf{g}_{\mathsf{L}_R,Q})+0,\sigma_{\mathsf{rand}}}$, $r \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{rand}}}$, $y \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}}$ and $\mathbf{v}$ is a vector of size ℓ_R of LWE samples of zero with noise parameter σ_R.

Hybrid 1. As Hybrid 0, but we set the message in $\mathbf{c}_{\mathsf{out}}$ to $f(m)$ instead of $\mathfrak{a}_{\mathsf{rot}} \cdot X^{\mathsf{Phase}(\mathbf{c})}[1]$. Assuming that $\mathsf{Error}(\mathbf{c}) \leq \frac{N}{t}$ and $\mathfrak{a}_{\mathsf{rot}}$ is such that $f(m) = \mathfrak{a}_{\mathsf{rot}} \cdot X^{\mathsf{Phase}(\mathbf{c})}[1]$ this change is only syntactical by correctness of the bootstrapping algorithm.

Hybrid 2. This hybrid is as Hybrid 1, but instead of computing $\mathbf{c}_{\mathsf{rand}} \leftarrow \mathbf{r}^\top \cdot \mathbf{v} = [\mathbf{a}_{\mathsf{rand}}, b_{\mathsf{rand}}]$, we take $\mathbf{c}_{\mathsf{rand}} = [\mathbf{a}_{\mathsf{rand}}, b_{\mathsf{rand}}]$ to be a fresh LWE encryption of zero. In particular, we take $\mathbf{a}_{\mathsf{rand}} \leftarrow_{\$} \mathbb{Z}_Q^N$ from the uniform distribution and take $\mathbf{b}_{\mathsf{rand}} = \langle \mathbf{a}_{\mathsf{rand}}, \mathbf{s} \rangle + e_{\mathsf{rand}}$ where $e_{\mathsf{rand}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{rand}} \cdot \sqrt{1+\ell_R \sigma_R^2}}$.

Claim. Given that

$$\sigma_{\mathsf{rand}} \geq \max\left(4\big((1-\gamma)(2\epsilon)^2\big)^{-1/\ell_R}, \sqrt{1+B_R} \cdot \max\left(||\mathbf{q}_{\mathsf{L}_R,Q}||, \sqrt{\mathsf{L}_R^2+1}\right) \cdot C_{\gamma,\ell_R}\right)$$

the statistical distance between Hybrid 2 and Hybrid 1 is at most $\epsilon + 2\gamma$ for some $\epsilon, \gamma > 0$.

Proof. Denote $\mathbf{v} = [\mathbf{a}_{\mathbf{v},i}, b_{\mathbf{v},i}]_{i=1}^{\ell_R}$ where $b_{\mathbf{v},i} = \langle \mathbf{a}_{\mathbf{v},i}, \mathbf{s}' \rangle + e_{\mathbf{v},i}$. Let $\widetilde{\mathbf{b}} = [b_{\mathbf{v},i}]_{i=1}^{\ell_R}$ and $\widetilde{\mathbf{e}} = [e_{\mathbf{v},i}]_{i=1}^{\ell_R}$. We can write $\widetilde{\mathbf{A}} = [\mathbf{a}_{\mathbf{v},i}, \ldots, \mathbf{a}_{\mathbf{v},\ell_R}] \in \mathbb{Z}_Q^{N \times \ell_R}$, $\mathbf{a}_{\mathsf{rand}} \leftarrow \widetilde{\mathbf{A}} \cdot \mathbf{r}$ and $b_{\mathsf{rand}} \leftarrow \widetilde{\mathbf{b}}^\top \cdot \mathbf{r} + r = \langle \mathbf{a}_{\mathsf{rand}}, \mathbf{s}' \rangle + \widetilde{\mathbf{e}}^\top \cdot \mathbf{r} + r$.

Now it is easy to see that the rest of the proof follows directly by applying Lemma 11. Note that the notation in Lemma 11 is mostly already in place, except that we set $m = \ell_R$ and $\mathsf{L} = \mathsf{L}_R$. Note that $w = 1$ for the lemma, because we sample $\mathbf{r}$ from $\mathcal{D}_{\Lambda^\perp_Q(\mathbf{g}_{\mathsf{L}_R,Q})+0,\sigma_{\mathsf{rand}}}$. Furthermore, the matrix $\mathbf{A}$ from the lemma is the matrix $\widetilde{A}$ in this hybrid and the error $\mathbf{e}$ from the lemma is the error $\widetilde{\mathbf{e}}$ in this hybrid.

Hybrid 3. This hybrid is as Hybrid 2, except that we choose $\mathbf{a}_{\mathsf{out}}$ from the uniform distribution. Note that both hybrids are in fact identical as from Hybrid 1, we have that $\mathbf{a}_{\mathsf{rand}}$ is sampled from the uniform distribution over $\mathbb{Z}_Q^N$, and we have $\mathbf{a}_{\mathsf{out}} = \mathbf{a}_{\mathsf{ext}} + \mathbf{a}_{\mathsf{rand}} \in \mathbb{Z}_Q^N$. This hybrid is only a syntactic change.

Hybrid 4. This hybrid is as hybrid 3 except that we compute $\mathbf{b}_{\mathsf{out}} = \langle \mathbf{a}_{\mathsf{out}}, \mathbf{s} \rangle + f(m) + e_{\mathsf{rand}} + e_{\mathsf{out}}$, where $e_{\mathsf{out}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}\sqrt{1+2nN\sigma_{\mathsf{br}}}}$. In particular, the noise term

is independent from the bootstrapped ciphertext $\mathbf{c}$ and the blind rotation key br.

Claim. Given that

$$\sigma_{\mathbf{x}} \geq \sqrt{1+B_{\mathsf{br}}} \cdot \max\left(||\mathbf{q}_{\mathsf{L}_{\mathsf{br}},Q}||, \sqrt{\mathsf{L}_{\mathsf{br}}^2+1}\right) \cdot C_{\delta,2\cdot n\cdot N\cdot \ell_{\mathsf{br}}}$$

the statistical distance between Hybrid 3 and Hybrid 4 is at most 2δ for some $\delta > 0$. The notation in Lemma 11 is mostly already in place, except we set $m = 2 \cdot n \cdot N \cdot \ell_{\mathsf{br}}$ and $\mathsf{L} = \mathsf{L}_{\mathsf{br}}$.

Proof. Note that we have

$$\mathsf{Error}(\mathbf{c}_{\mathsf{ext}}) = \sum_{i=1}^{n}\sum_{j=1}^{2\ell_{\mathsf{br}}}\sum_{k=1}^{N} \mathbf{e}_{i,j}[k] \cdot \mathfrak{x}_{i,j}[k] = \sum_{i=1}^{n}\sum_{k=1}^{N}\sum_{j=1}^{2\ell_{\mathsf{br}}} \mathbf{e}_{i,j}[k] \cdot \mathfrak{x}_{i,j}[k].$$

We will group the terms $\mathbf{e}_{i,j}[k] \cdot \mathfrak{x}_{i,j}[k]$ into vectors by the j iterator. We write $\widehat{\mathbf{e}_{i,k}} = \left[\mathbf{e}_{i,j}[k]\right]_{j=1}^{2\ell_{\mathsf{br}}}$ and $\widehat{\mathbf{x}_{i,k}} = \left[\mathfrak{x}_{i,j}[k]\right]_{j=1}^{2\ell_{\mathsf{br}}}$. Note that $\widehat{\mathbf{x}_{i,k}} \in \mathcal{D}_{\Lambda_Q^{\perp}(\mathbf{G}_{\mathsf{L}_{\mathsf{br}},2})+\mathsf{G}_{\mathsf{det}}^{-1}(\mathbf{c}_{\mathsf{acc},i}[k]),\sigma_{\mathbf{x}}}$. In other words $\widehat{\mathbf{x}_{i,k}}$ is the Gaussian sampling of the kth coefficient (where $k \in [2\ell_{\mathsf{br}}]$, meaning that we take the concatenation of the two polynomials in the accumulator $\mathbf{c}_{\mathsf{acc},i}$) in the ith iteration of the blind rotation algorithm. Now we can write

$$\mathsf{Error}(\mathbf{c}_{\mathsf{ext}}) = e_{\mathsf{rand}} + \mathsf{Error}(\mathbf{c}_{\mathsf{ext}}) + y = e_{\mathsf{rand}} + y + \sum_{i=1}^{n}\sum_{k=1}^{N} \widehat{\mathbf{e}_{i,k}}^{\top} \cdot \widehat{\mathbf{x}_{i,k}}.$$

We can further represent $\mathsf{Error}(\mathbf{c}_{\mathsf{ext}})$ as a product $\mathbf{x}^{\top} \cdot \mathbf{e}$ of two vectors $\mathbf{e} = [\widehat{\mathbf{e}_{i,k}}^{\top}]_{i=1,k=1}^{n,N}$ and $\mathbf{x} = [\widehat{\mathbf{x}_{i,k}}]_{i=1,k=1}^{n,N}$. Note that

$$\mathbf{x} \in \mathcal{D}_{\Lambda_Q^{\perp}(\mathbf{G}_{\mathsf{L}_{\mathsf{br}},Q,2\cdot n\cdot N})+\mathsf{G}_{\mathsf{det}}^{-1}([\mathbf{c}_{\mathsf{acc},i}[k]]_{i=1,k=1}^{n,N}),\sigma_{\mathbf{x}}}$$

since it is just a concatenation of $n \cdot N$ vectors from $\mathcal{D}_{\Lambda_Q^{\perp}(\mathbf{G}_{\mathsf{L}_{\mathsf{br}},2})+\mathsf{G}_{\mathsf{det}}^{-1}(\mathbf{c}_{\mathsf{acc},i}[k]),\sigma_{\mathbf{x}}}$. Finally, note that $\mathbf{a}_{\mathsf{out}}$ and the message are independent of $\mathsf{Error}(\mathbf{c}_{\mathsf{ext}})$. Therefore we can apply Lemma 9 to $\mathbf{x}^{\top} \cdot \mathbf{e} + y$ with $m = 2 \cdot n \cdot N \cdot \ell_{\mathsf{br}}$ and $\mathsf{L} = \mathsf{L}_{\mathsf{br}}$.

Finally, we have that $\mathbf{c}_{\mathsf{out}}$ is distributed as in the theorem statement, and is in particular independent of the input ciphertext $\mathbf{c}$ and bootstrapping key br.

Remark 2 (On FHEW Blind Rotation.). Theorem 1 works for the blind rotation algorithm from [21]. We chose to focus on this blind rotation algorithm as it is currently faster in practice for binary and ternary LWE keys. We note, however, that it is even easier to prove an analogical theorem with the blind rotation from FHEW [29] because it is a sequence of external products. Consequently, we can omit the "Single MUX Gate" step in the proof of Theorem 1.

Proof of Circuit Privacy. Remind that according to Definition 2, to prove circuit privacy we have to show a simulator that, on input $m_{\mathsf{out}} = \mathcal{C}(m_1, \ldots, m_n)$ outputs a ciphertexts of m_{out} that is distributed as an output of the Eval algorithm when evaluating $\mathcal{C}$ on encryptions of $m_1, \ldots, m_n$. We build such simulator by sampling an encryption of zero using the vector $\mathbf{v}$ of LWE samples of zero form the evaluation key, adding m_{out} and the missing noise term e_{out} that would steam from blind rotation. Circuit privacy then follows from Theorem 1.

Theorem 2. *Let $\mathcal{C}$ be a polynomial size circuit and $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Eval}(\mathsf{ek}, [\mathsf{ct}_i]_{i=1}^n, \mathcal{C})$, where $\mathsf{ct}_i \leftarrow \mathsf{Dec}(\mathsf{sk}, m_i)$ and Eval is as described in Section 3, where the bootstrapping algorithm for of the output gate is set to $\mathsf{ver} = \mathsf{simul}$. If the parameters of the FHE scheme are chosen such that $\Delta(\mathbf{c}_{\mathsf{out}}, \mathbf{c}_{\mathsf{fresh}}) \leq \mathsf{negl}(\lambda)$, where $\mathbf{c}_{\mathsf{fresh}}$ is as in Theorem 1, then the evaluation process is circuit private.*

Proof. To show circuit privacy we need to show a simulator Sim that gets as input ek and m_{out}. The proof follows nearly immediately, from Theorem 1. The simulator samples $\mathbf{r} \leftarrow_{\$} \mathcal{D}_{\Lambda_Q^\perp(\mathbf{G}_{\lfloor R})+0,\sigma_{\mathsf{rand}}}$, $r \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{rand}}}$ and $e_{\mathsf{out}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}\sqrt{1+2nN\sigma_{\mathsf{br}}}}$. Then Sim computes $\mathbf{c}_{\mathsf{rand}} \leftarrow \mathbf{r}^\top \cdot \mathbf{v} + r$ and $\mathbf{c}_{\mathsf{fresh}} \leftarrow \mathbf{c}_{\mathsf{rand}} + e_{\mathsf{out}} + m_{\mathsf{out}}$. Denote $\mathbf{c}_{\mathsf{rand}} = [\mathbf{a}_{\mathsf{rand}}, b_{\mathsf{rand}}]$. First we argue that $\mathsf{Error}(\mathbf{c}_{\mathsf{rand}}) \in \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{rand}}\cdot\sqrt{1+\ell_R\sigma_R^2}}$ and $\mathbf{a}_{\mathsf{rand}} \in \mathbb{Z}_Q^N$ is close to the uniform distribution, as in the proof of Theorem 1 (Hybrid 1). We end up with a LWE sample $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$ where $b_{\mathsf{fresh}} = \langle \mathbf{a}_{\mathsf{fresh}}, \mathbf{s}' \rangle + f(m) + e_{\mathsf{rand}} + e_{\mathsf{out}}$. The statistical distance between $\mathbf{c}_{\mathsf{out}}$ and $\mathbf{c}_{\mathsf{fresh}}$ output by Sim follows now directly from Theorem 1.

Remark 3 (Removing the need for Lemma 11). Note that for the proof of Theorem 1, we need to invoke Lemma 11 only to show that the ciphertext $\mathbf{c}_{\mathsf{rand}}$ is distributed like a "fresh" ciphertext. Furthermore, note that we only need to same this one ciphertext one time per bootstrapping after the blind rotation in order to mask the $\mathbf{a}_{\mathsf{ext}}$ vector from $\mathbf{c}_{\mathsf{ext}}$ with the uniform random $\mathbf{a}_{\mathsf{rand}}$ vector from $\mathbf{c}_{\mathsf{rand}}$. An alternative way to sample $\mathbf{c}_{\mathsf{rand}}$ is to publish the $\mathbf{v}$ key with respect to a larger modulus, given that the RLWE problem remains secure, use the standard linear-combination with noise flooding technique, and modulus switch to Q. In this case, we only need to modify Hybrid 2, and show the different distribution of $\mathbf{c}_{\mathsf{rand}}$. Furthermore, we need to modify the simulator in Theorem 2 and sample $\mathbf{c}_{\mathsf{rand}}$ accordingly.

5 Parameters, Implementation, and Experiments

In this section, we discuss our parameter choices, implementation, and experiments for our method as well as for the washing machine method by Ducas and Stehlé [30]. Remind that Ducas and Stehlé [30] left finding correct parameters as an open problem. In this section, we address this problem and rule out the possibility of instantiating FHEW/TFHE negligible statistical security over low-degree rings that are often used for efficiency. We give a comparison of both methods when parameters are targeted towards an implementation over integers modulo an NTT-friendly prime number and over integers modulo a power of two represented as double-precision floating point numbers.

5.1 Parameters.

We choose our parameter sets to target 128-bit security for the (R)LWE samples and 80-bits statistical security when running the bootstrapping in simul mode. The parameters are listed in Table 1. We estimate the (R)LWE security using the latest commit of the LWE estimator [6]. We wrote a python script that we published alongside our source code to estimate the statistical security. We choose similar parameters to make a good comparison between our method and the Ducas-Stehlé washing machine method [30] that we refer to as DS-WM. For completeness we recall the relevant lemmas from [30] that we used for our estimations in Supplementary Material C. For all parameters we chose the same ring dimension $N = 2^{11}$. The strategy is to choose the highest modulus such that: (1) the RLWE problem remains 128-bit secure according to the LWE estimator [6], (2) the modulus is below 50-bits for Ours-Int and DS-WM-Int to allow for faster multiplication of ring elements, and (3) computing convolutions does not introduce significant numerical errors. Furthermore, we choose the LWE parameters for the key switching key and the masking key $\mathbf{v}$, the same for all solutions. Then we choose the decomposition bases. We chose the highest decomposition base that: (1) gives us required correctness, (2) maximizes L_{Bk}-simul, (3) L_{Bk}-det $= \mathsf{L}_{\mathsf{Bk}}$-simul^k for some $k \geq 1$ for deterministic gadget decomposition gives us the required correctness as well. Note that the last condition allows us to significantly optimize computation when using deterministic gadget decomposition at no additional cost to the key size. This is because, when using deterministic gadget decomposition we must only use every k-th RLWE ciphertext of the gadget ciphertext when computing the multisum in the external product. In other words, computing the external product requires roughly $k\times$ fewer ring multiplications. Finally, we compute the noise parameters for the Gaussian sampling and noise flooding given the desired security level and all other parameters.

log$_2$(Param.) / Method	Q	L_{Bk}, ℓ_{Bk}-simul	L_{Bk}, ℓ_{Bk}-det	σ_{Ksk}	$\sigma_{\mathbf{x}}$	U_x	σ_{rand}	Δ
Ours-Int	48	8, 6	24, 2	26	17.7	-	12.3	80
DS-WM-Int	48	5,10	25,2	26	-	42.6	12.3	20
Ours-Double	36	4, 9	12, 3	14	8.9	-	15.8	80
DS-WM-Double	36	4, 9	12, 3	14	-	30.8	15.8	8

Table 1. Parameter Sets. We list base-two logarithms of the parameters. For all sets we set $n = 912$, $N = 2^{11}$, $\mathsf{L}_R = 2^3$, $\mathsf{L}_{\mathsf{Ksk}} = 2^7$ and $\sigma_{\mathsf{br}} = 3.2$. Note that the column L_{Bk}-simul gives the decomposition base for ver = simul and L_{Bk}-det for ver = det. The column U_x refers to the uniform distribution interval for the noise flooding in DS-WM. The column Δ refers to the statistical distance from a random ciphertext after a single bootstrapping operation. Consequently, for DS-WM-Int and DS-WM-Double, we have to run the bootstrapping 4 and 10 times, respectively.

As we can read from Table 1, we managed to find parameters where Ours-Int method needs $7 = \ell_{\mathsf{br}} + 1$ ring multiplications per gadget multiplication in simul mode, whereas DS-WM-Int requires $11 = \ell_{\mathsf{br}} + 1$. We stress that the noise flooding method is already incorrect for a decomposition base $\mathsf{L}_{\mathsf{br}} = 2^6$ meaning that this is the best base choice for performance. Nevertheless, for the flooding set, we can use a similar decomposition basis in det mode as in our set resulting in the same efficiency for deterministic bootstrapping. This parameter choice allows us to compare both methods better as we can now focus solely on discussing the differences between the sets in simul mode. For the det mode efficiency of the sets is the same and correctness is very similar. In the case, where the modulus is a power of L_{br} and implementation uses double precision floating point arithmetic to compute polynomial multiplication, we need a much lower modulus, and decomposition bases. In this case we found sets where all decomposition parameters are the same for our method on for DS-WM.

To estimate correctness, we compute the error function which is defined as $\mathsf{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$, and which given standard deviation $1/\sqrt{2}$ returns the probability that a Gaussian distributed random variable with mean 0 lies within the interval $[-x, x]$. Furthermore, we define $\mathsf{erfc}(x) = 1 - \mathsf{erf}(x)$. Our estimator must deal with random variables of mean $\neq 0$ and standard deviations other than $1/\sqrt{2}$. Hence given mean c and standard deviation σ we compute $\mathsf{erfc}(\frac{x-c}{\sigma \cdot \sqrt{2}})$, where $x = \frac{Q}{2 \cdot t}$ is the interval given by the modulus Q and message space modulus t to obtain the probability that our noise exceeds the tolerable bound and "shifts" the plaintext. In Table 2, we list the probabilities of having an error while bootstrapping. The errors are given as base-two logarithms for readability. As we may see, our method has a different characteristic when it comes to correctness than the DS-WM. First observe that correctness of $\mathbf{c}_{\mathsf{in}}$ is lower than correctness of $\mathbf{c}_{\mathsf{out}}$. This is due to the modulus switching to a much smaller modulus $2N$ and the rounding error. Then note that when increasing the message space, our method is still correct for $\mathbf{c}_{\mathsf{out}}$, while DS-WM's correctness collapses already at $t = 5$. This is the consequence of needing to run the bootstrapping step numerous times to sanitize a ciphertext. However, since our method requires only a single bootstrapping invocation, we can output ciphertexts of larger precision.

Instantiation over degree $N = 2^{10}$ Rings. Numerous works like [29, 21, 22] choose parameters for the ring $\mathbb{Z}_Q[X]/(X^N + 1)$ setting the degree to $N = 2^{10}$. The obvious benefit is that the timings are fast, while the ring dimension is high enough to give 128-security according to the LWE estimator [6]. What is important is that when assessing correctness, these works often report only on the correctness $\mathbf{c}_{\mathsf{out}}$ and ignore the correctness of $\mathbf{c}_{\mathsf{in}}$. Notably, Ducas and Stehlé [30] propose to instantiate their washing machine method on a parameter set from the Ducas and Micciancio's FHEW bootstrapping [29], albeit they note that their instantiation is heuristic and leave a serious analysis as an open problem. We investigated the possibility of choosing a parameter set for the rings of dimension $N = 2^{10}$, and, unfortunately, we found it to be infeasible. We chose the highest modulus possible. Furthermore, we took the decomposition bases to be equal to 2 (smallest possible) to maximize correctness. Even when decreasing the

	Ours-Int			
t	det		simul	
	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$
4	−849	−197	−494	−129
5	−545	−82	−317	−54
6	−380	−33	−221	−22
⋮				
10	−136	0.99	−82	0
11	−113	0.99	−68	0

	Ducas-Stehlé: DS-WM-Int			
t	det		simul	
	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$
4	−785	−186	−85	−85
5	−504	−78	−18	−18
6	−351	−31	−1.25	−1.25
⋮				
10	−128	0	0	0
11	−107	0	0	0

	Ours-Double			
t	det		simul	
	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$
4	−956	−215	−700	−170
5	−613	−89	−449	−71
6	−427	−36	−313	−29
⋮				
10	−156	0.88	−115	0.94
11	−129	0.99	−95	0.98

	Ducas-Stehlé: DS-WM-Double			
t	det		simul	
	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$
4	−956	−215	−83	−83
5	−613	−89	−15	−15
6	−427	−36	−0.07	−0.07
⋮				
10	−156	0.88	0	0
11	−129	0.99	0	0

Table 2. Correctness Estimates. We give the probability of failure to correctly decrypt $\mathbf{c}_{\mathsf{out}}$ and $\mathbf{c}_{\mathsf{in}}$ for a given message space t. We give the correctness estimates as base-two logarithm. We mark failure probabilities below 2^{-80} with green, and above that threshold with red.

LWE dimension n and noise parameter to insecure levels, we noticed that, while correctness of $\mathbf{c}_{\mathsf{out}}$ was satisfactory, correctness of $\mathbf{c}_{\mathsf{in}}$ was below the 2^{-30} level. The reason for this is the rounding error when modulus switching from Q to $2N$. In other words, the ring degree is already so small that ciphertexts modulo $2N$ cannot accommodate the rounding error within the interval N/t. Note that we use binary keys here, which is even more beneficial for correctness, than if we would use ternary or Gaussian distributed keys as in FHEW-style schemes [29]. To conclude, our analysis shows that the parameter choice in [30] for DS-WM cannot give circuit privacy better than 30-bits, and we need to instantiate the method with a larger ring. Furthermore, we also rule out the possibility of running parameters from [21, 22] in simulation mode due to the small ring degree with statistical security larger than 30-bits.

	Total [s]		BL [s]		$\mathsf{G}^{-1}_{\mathsf{simul}}$	Ksk [MB]		Bk [MB]		$\mathbf{v}$ [MB]	
	det	simul	det	simul		8-bit	64-bit	8-bit	64-bit	8-bit	64-bit
Ours-Int	0.14	1.44	0.14	1.44	78%	79	104	134	179	0.12	0.52
DS-WM-Int	0.14	2.72	0.14	0.71	–			224	298		
Ours-Double	0.27	1.22	0.27	1.22	42%	56	89	168	268	0.24	0.39
DS-WM–Double	0.27	6.38	0.27	0.63	–			168	268		

Table 3. Performance. The BL and KS columns give the blind rotation and key switching timings. In the "Total" column we give the over time to run a bootstrapping operation. The $\mathsf{G}^{-1}_{\mathsf{simul}}$ column represents the proportion of the Gaussian sampling in the total computation. Remind that in the DS-WM, we must run bootstrapping several times. The Ksk, Bk, and $\mathbf{v}$ columns give sizes of the respective public keys. We list the size of public keys counted by storing an integer/float in an array of 8-bit or storing an integer/float in a 64-bit register.

5.2 Implementation and Performance.

We implemented the schemes in C++11 and tested it on a machine with 11th Gen Intel(R) Core(TM) i7-11850H 2.50GHz processor that supports AVX2 and AVX-512 instructions. The timing results and the size of the evaluation keys for the bootstrapping algorithms are given in Figure 3. We omit including the timing for masking with the vector $\mathbf{v}$, as it is negligible in comparison to other timings (for our parameter sets, it takes only 16 scalar multiplication and additions of LWE ciphertexts). To implement negacyclic convolutions for Ours-Int and DS-WM-Int, we used the Intel Hexl library [11]. The library gives a high-performance implementation of Number Theoretic Transforms optimized for the ring $\mathbb{Z}_Q[X]/(X^N+1)$ and takes full advantage of Intel AVX instructions. To compute the negacyclic convolutions for Ours-Double and DS-WM-Double we use the `FFTW` library [31] that implements fast Fourier transforms on IEEE-754 double precision floating point arithmetic.

For the Gaussian sampling algorithm, we implemented two methods. For the case where the modulus is a power of L_{Bk} we implement the simple and very efficient Gaussian sampler from [56]. For general moduli, which is the case for Our-Int and DM-WM-Int, we implement the method by Genise, and Micciancio [32]. To sample from the discrete Gaussian distribution, we used the Gaussian sampler from the C++-standard library with Mersenne Twister, with rounding to the closest integer. Note that the rounding may distort the distribution. However, in our case, we use a standard deviation of about $2^{8.9}$ and $2^{17.7}$ which is quite high. Hence the resulting distribution should be statistically close to a discrete Gaussian.

As we can see from Table 3, our method is faster when compared to both DS-WM methods. In particular, Ours-Int is roughly 1.88× faster than DS-WM-Int and has a 1.8× smaller blind rotation key. Then, Our-Double is about 5.22× faster than its DS-WM-Double analog. Note that Ours-Int has a smaller blind rotation key than its DS-WM-Int counterpart, which is due to smaller error growth when increasing the L_{Bk} parameter. Ours and DS-WM have the same computation time for a deterministic bootstrapping operation in their respective implementations. This is a result of our specific parameter choice.

The "Double" parameters are slower than the "Int" parameters in det mode. The reason for this is that the "Double" parameters require computing more negacyclic convolutions than the "Int" parameters due to their different modulus and decomposition factors. It's important to note that all parameter sets should achieve similar correctness levels. Therefore, for the "Double" parameter sets, which are intended to be implemented using IEEE-754 double precision floating point format, we are constrained by the precision of the arithmetic.

When estimating the key size, we list two methods. One assumes storing integers or (discretized) floating point numbers in an array of 8-bit bytes. This may potentially give an edge to the "Double" parameter sets since these sets use a smaller modulus. However, these sets still require larger key material because we need to store mode ring elements. Although the byte array representation can be used when transmitting a key, when computing the bootstrapping operation, the integers or floats are stored in random access memory in 64-bit registers, regardless of whether the modulus is a 48-bit or 36-bit number. In any case the Our-Int parameter set outperforms all other sets. The size of a ciphertext is the same for every parameter set. Note that in all sets we can send the first ciphertexts with a modulus equal to $2 \cdot N = 2^{12}$, and all sets have the same LWE dimension $n = 912$. Consequently, the ciphertexts will take approximately 0.46 [MB].

Remark 4 (On Efficiency of the DS-WM-Int Set). Note that in the DS-WM-Int parameter set, we allowed parameters that require about 1.67× larger blind rotation key. If we would make a comparison between schemes with the same key size in the "Int" category, which is the only fair comparison, then we need to set the same decomposition bases in DS-WM-Int as in Ours-Int. For such base, DS-WM-Int with flooding noise achieving 20-bits of statistical distance has correctness only around 2^{30}. To remedy for a smaller flooding noise, we need

to perform five bootstrapping repetitions instead of four. Consequently, all key sizes and deterministic computation time are the same as for Ours-Int, but the total time for sanitizing a ciphertext with DS-WM-Int would be 3.55 seconds in simul mode instead of 2.72 seconds as in Table 3.

Finally, note that in the case of Our-Int, sanitization is roughly 10× slower than deterministic computation. For Our-Double, sanitization is only 2× slower than deterministic computation. Partially, the reason for this is that the sanitization algorithms have smaller decomposition bases, but a notable portion of the computation is spent on computing Gaussian sampling. For Gaussian sampling, in the special case where the modulus is a power of the decomposition basis, as is the case in Our-Double, approximately half of the computation is spent on sampling Gaussians. In the general case, such as Our-Int, where the modulus is an NTT-friendly prime number, Gaussian sampling constitutes approximately 78% of the entire computation. We stress that the Gaussian sampling step is implemented in a straightforward manner. There is still much room for improvement in the implementation. In particular, the method can be parallelized over the $N = 2^{11}$ coefficients of the ring. Similarly, as with Intel Hexl for NTT, an optimized implementation could take advantage of AVX vector processor extensions to parallelize parts or even the entire sampling algorithm. We do not see much room for improvement in the washing machine method because its only difference from a deterministic bootstrap is the choice of uniform flooding noise.

6 Conclusions and Open Problems

We showed that it is practically feasible to build an efficient FHE scheme with circuit privacy, that is competitive with the Ducas-Stehlé washing machine method [30]. Importantly, the major bottleneck in our implementation is Gaussian sampling. We believe that an optimized implementation of gaussian sampling exploiting AVX vector extentions like in Intel Hexl would greatly the improve performance. Finally, an interesting problem is to analyze whether we can use randomized gadget decomposition that has output from other distributions and discrete Gaussian.

References

1. PALISADE Lattice Cryptography Library (release 1.11.5). https://palisade-crypto.org/ (2021)
2. Lattigo v4. Online: https://github.com/tuneinsight/lattigo (Aug 2022), ePFL-LDS, Tune Insight SA
3. Abspoel, M., Bouman, N.J., Schoenmakers, B., de Vreede, N.: Fast secure comparison for medium-sized integers and its application in binarized neural networks. In: Matsui, M. (ed.) Topics in Cryptology – CT-RSA 2019. Lecture Notes in Computer Science, vol. 11405, pp. 453–472. Springer, Heidelberg (Mar 2019). https://doi.org/10.1007/978-3-030-12612-4_23

4. Aggarwal, D., Regev, O.: A note on discrete gaussian combinations of lattice vectors (2013). https://doi.org/10.48550/ARXIV.1308.2405, https://arxiv.org/abs/1308.2405
5. Agrawal, S., Gentry, C., Halevi, S., Sahai, A.: Discrete Gaussian leftover hash lemma over infinite domains. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Part I. Lecture Notes in Computer Science, vol. 8269, pp. 97–116. Springer, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42033-7_6
6. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/doi:10.1515/jmc-2015-0016, https://doi.org/10.1515/jmc-2015-0016
7. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 1–20. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_1
8. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 297–314. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_17
9. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 483–501. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_29
10. Blatt, M., Gusev, A., Polyakov, Y., Goldwasser, S.: Secure large-scale genome-wide association studies using homomorphic encryption. Proceedings of the National Academy of Sciences **117**(21), 11608–11613 (2020)
11. Boemer, F., Kim, S., Seifu, G., de Souza, F.D., Gopal, V., et al.: Intel HEXL (release 1.2). https://github.com/intel/hexl (2021)
12. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758 (2018), https://eprint.iacr.org/2018/758
13. Bourse, F., Del Pino, R., Minelli, M., Wee, H.: Fhe circuit privacy almost for free. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 62–89. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
14. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science. pp. 309–325. Association for Computing Machinery (Jan 2012). https://doi.org/10.1145/2090236.2090262
15. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd Annual Symposium on Foundations of Computer Science. pp. 97–106. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.12
16. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science. pp. 1–12. Association for Computing Machinery (Jan 2014). https://doi.org/10.1145/2554797.2554799
17. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035 (2017), https://eprint.iacr.org/2017/035

18. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 395–412. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363207
19. Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 315–337. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_12
20. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 1243–1255. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134061
21. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 3–33. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
22. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (Jan 2020). https://doi.org/10.1007/s00145-019-09319-x
23. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
24. Chillotti, I., Joye, M., Ligier, D., Orfila, J.B., Tap, S.: Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In: WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. vol. 15 (2020)
25. Chongchitmate, W., Ostrovsky, R.: Circuit-private multi-key FHE. In: Fehr, S. (ed.) PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II. Lecture Notes in Computer Science, vol. 10175, pp. 241–270. Springer, Heidelberg (Mar 2017). https://doi.org/10.1007/978-3-662-54388-7_9
26. Dachman-Soled, D., Gong, H., Kulkarni, M., Shahverdi, A.: Towards a ring analogue of the leftover hash lemma. Journal of Mathematical Cryptology **15**(1), 87–110 (2021). https://doi.org/doi:10.1515/jmc-2020-0076, https://doi.org/10.1515/jmc-2020-0076
27. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology – EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 523–540. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_31
28. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. p. 201–210. ICML'16, JMLR.org (2016)
29. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
30. Ducas, L., Stehlé, D.: Sanitization of FHE ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I. Lecture Notes

in Computer Science, vol. 9665, pp. 294–310. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49890-3_12
31. Frigo, M., Johnson, S.G.: Fftw. https://www.fftw.org (2021)
32. Genise, N., Micciancio, D.: Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 174–203. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_7
33. Genise, N., Micciancio, D., Polyakov, Y.: Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 655–684. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17656-3_23
34. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford, CA, USA (2009)
35. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
36. Gentry, C., Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd Annual Symposium on Foundations of Computer Science. pp. 107–109. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.94
37. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 7293, pp. 1–16. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_1
38. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop homomorphic encryption and rerandomizable Yao circuits. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. Lecture Notes in Computer Science, vol. 6223, pp. 155–172. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7_9
39. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th Annual ACM Symposium on Theory of Computing. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
40. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
41. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 641–670. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_25
42. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021). https://doi.org/10.1007/s00145-020-09368-7
43. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce. p. 78–86. EC '99, Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/336992.337012, https://doi.org/10.1145/336992.337012

44. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007: 4th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 4392, pp. 575–594. Springer, Heidelberg (Feb 2007). https://doi.org/10.1007/978-3-540-70936-7_31
45. Jiang, X., Kim, M., Lauter, K.E., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security. pp. 1209–1222. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243837
46. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018: 27th USENIX Security Symposium. pp. 1651–1669. USENIX Association (Aug 2018)
47. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**(4), 23–31 (2018)
48. Kim, D., Son, Y., Kim, D., Kim, A., Hong, S., Cheon, J.H.: Privacy-preserving approximate gwas computation based on homomorphic encryption. BMC Medical Genomics **13**(7), 1–12 (2020)
49. Kluczniak, K.: Ntru-ν-um: Secure fully homomorphic encryption from ntru with small modulus. Cryptology ePrint Archive, Paper 2022/089 (2022), https://eprint.iacr.org/2022/089, https://eprint.iacr.org/2022/089
50. Kluczniak, K., Schild, L.: FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/1135 (2021), https://eprint.iacr.org/2021/1135
51. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 619–631. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134056
52. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. Cryptology ePrint Archive, Report 2021/1337 (2021), https://eprint.iacr.org/2021/1337
53. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 35–54. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_3
54. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy. pp. 134–134 (1986). https://doi.org/10.1109/SP.1986.10022
55. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. In: 45th Annual IEEE Symposium on Foundations of Computer Science. pp. 372–381 (2004). https://doi.org/10.1109/FOCS.2004.72
56. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 700–718. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_41
57. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like Cryptosystems, p. 17–28. Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3474366.3486924

58. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
59. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E., Koushanfar, F.: XONN: XNOR-based oblivious deep neural network inference. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019: 28th USENIX Security Symposium. pp. 1501–1518. USENIX Association (Aug 2019)
60. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. foundations of secure computation (1978), 169–180. Search in (1978)
61. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 27–47. Springer, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20465-4_4
62. Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347 (2021), https://eprint.iacr.org/2021/1347

Appendices

A Error Analysis and Missing Algorithms

In this section we give the noise analysis and correctness proofs.

Proof. (External Product, Lemma 2). Denote $\mathbf{x} = \mathsf{G}^{-1}_{\mathsf{ver}}(\mathbf{c}, \mathsf{L}_{\mathsf{br}})$ and $\mathbf{c} = [\mathfrak{b}, \mathfrak{a}]$. Then we compute

$$\begin{aligned}\mathbf{C}_G \cdot \mathbf{x} =& \sum_{i=1}^{\ell_{\mathsf{br}}} \mathsf{RLWE}_{\sigma_G}(\mathfrak{s}, \mathfrak{m}_G \cdot \mathsf{L}^{i-1}_{\mathsf{br}}) \cdot \mathbf{x}[i] \\ &+ \sum_{i=\ell_{\mathsf{br}}+1}^{2\ell_{\mathsf{br}}} \mathsf{RLWE}_{\sigma}(\mathfrak{s}, -\mathfrak{s} \cdot \mathfrak{m}_G \cdot \mathsf{L}^{i-1}_{\mathsf{br}}) \cdot \mathbf{x}[i] \\ =& \mathsf{RLWE}_{\sigma}(\mathfrak{s}, \mathfrak{m}_G \cdot \mathfrak{b}) + \mathsf{RLWE}_{\sigma}(\mathfrak{s}, -\mathfrak{s} \cdot \mathfrak{m}_G \cdot \mathfrak{a}) \\ =& \mathsf{RLWE}_{\sigma_1}(\mathfrak{s}, \mathfrak{m}_G \cdot (\mathfrak{m} + \mathfrak{e})) \\ =& \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathfrak{s}, \mathfrak{m}_G \cdot \mathfrak{m})).\end{aligned}$$

The following $\sigma_1^2 \leq 2\ell_{\mathsf{br}} \cdot N \cdot \sigma_G^2 \cdot \mathsf{B}(\mathsf{G}^{-1}_{\mathsf{ver}}(., \mathsf{L}_{\mathsf{br}}))$ holds because we compute the multisum $\sum_{i=1}^{2\ell_{\mathsf{br}}} \cdot \mathfrak{e}_i \cdot \mathbf{x}[i]$ where $\mathfrak{e}_i$ is the error of the ith RLWE sample in $\mathbf{C}_G$. Note that each coefficient of $\mathfrak{e}_i \cdot \mathbf{x}[i]$ in the ring $\mathcal{R}_Q$ is a negacyclic convolution of the coefficients in $\mathfrak{e}_i$ and $\mathbf{x}[i]$. Finally, $\sigma^2_{\mathsf{out}} \leq \sigma^2 + \mathfrak{m}_G \cdot \sigma_1^2$ holds because, $\mathfrak{m}_G \cdot \mathfrak{e}$ is 0 or $\mathfrak{e}$ depending on the bit $\mathfrak{m}_G$.

Proof. (Mux Gate, Lemma 3). Note that the RGSW sample $\mathbf{C}$ encodes a bit $\mathsf{m}_{\mathbf{c}} \in \{0, 1\}$. As in the proof of Lemma 2 we have $\mathbf{c}_{\mathsf{out}} = \mathsf{RLWE}_{\sigma_1}(\mathfrak{s}, \mathsf{m}_{\mathbf{c}} \cdot (\mathsf{m}_{\mathbf{d}} - \mathsf{m}_{\mathbf{h}} + \mathfrak{e}_{\mathbf{d}} - \mathfrak{e}_{\mathbf{d}}) + \mathbf{h}$. Hence, for $\mathsf{m}_{\mathbf{C}} = 0$, we get

$$\mathbf{c}_{\mathsf{out}} = \mathsf{RLWE}_{\sigma_1}(\mathfrak{s}, \mathsf{m}_{\mathbf{h}} + \mathfrak{e}_{\mathbf{h}}) = \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathfrak{s}, \mathsf{m}_{\mathbf{h}}),$$

and for $\mathsf{m}_{\mathbf{C}} = 1$, we get

$$\mathbf{c}_{\mathsf{out}} = \mathsf{RLWE}_{\sigma_1}(\mathfrak{s}, \mathsf{m}_{\mathbf{d}} + \mathfrak{e}_{\mathbf{g}}) = \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathfrak{s}, \mathsf{m}_{\mathbf{d}}).$$

In either case, we have that $\sigma^2_{\mathsf{out}} \leq 2\ell_{\mathsf{br}} \cdot N \cdot \sigma_G^2 \cdot \mathsf{Var}(\mathsf{G}^{-1}_{\mathsf{ver}}(., \mathsf{L}_{\mathsf{br}})) + \sigma^2$, because as we assumed the noise parameter for $\mathbf{d}$ and $\mathbf{h}$ is the same.

Proof. (Modulus Switching, Lemma 4). Denote $\mathbf{c} = (b, \mathbf{a})$, where $b = \mathbf{a}^\top \cdot \mathbf{s} + \mathsf{m} + e \in \mathbb{Z}_Q$ where e has variance σ^2. Then we have the following:

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

where $r \in \mathbb{R}$ and $\mathbf{r} \in \mathbb{R}^n$ are in $[-\frac{1}{2}, \frac{1}{2}]$. Then we have

$$\begin{aligned}\sigma^2_{\mathsf{out}} &= \mathsf{Var}(\frac{q}{Q} \cdot e + r + \mathbf{r}^\top \cdot \mathbf{s}) \\ &= \mathsf{Var}(\frac{q}{Q} \cdot e) + \mathsf{Var}(\mathbf{r}^\top \cdot \mathbf{s})) \\ &= \frac{q^2}{Q^2} \cdot \sigma^2 + \sum_{i=1}^{n} \mathsf{Var}(\mathbf{r}[i] \cdot \mathbf{s}[i]) \\ &\leq \frac{q^2}{Q^2} \cdot \sigma^2 + \frac{1}{4} \cdot \mathsf{Ha}(\mathbf{s}) \cdot \mathsf{Var}(\mathbf{s}).\end{aligned}$$

The expectation of the output noise satisfies

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

given that the expectation of e is 0.

Proof. (Sample Extraction, Lemma 5). Denote $\mathbf{s} = \mathfrak{s} \in \mathbb{Z}_Q^N$ and $b = \mathfrak{b}[1] \in \mathbb{Z}_Q$. Denote $\mathfrak{b} = \mathfrak{a} \cdot \mathfrak{s} + \mathfrak{m} + \mathfrak{e} \in \mathcal{R}_Q$, $\mathfrak{m} = \sum_{i=1}^{N} \mathfrak{m}[i] \cdot X^{i-1}$ and $\mathfrak{e} = \sum_{i=1}^{N} \mathfrak{e}[i] \cdot X^{i-1}$ then it is easy to see, that $b = (\mathfrak{a} \cdot \mathfrak{s})[1] + \mathfrak{m}[1] + \mathfrak{e}[1]$. Furthermore, denote $\mathfrak{a} = \sum_{i=1}^{N} \mathfrak{a}[i] \cdot X^{i-1}$ and $\mathfrak{s} = \sum_{i=1}^{N} \mathfrak{s}[i] \cdot X^{i-1}$. Denote $\mathfrak{s} \cdot \mathfrak{a} = (\sum_{i=1}^{N} \mathfrak{a}[i] \cdot X^{i-1}) \cdot (\sum_{i=1}^{N} \mathfrak{s}[i] \cdot X^{i-1})$. By expanding the product we have that the constant coefficient is given by $(\mathfrak{s} \cdot \mathfrak{a})[1] = \mathfrak{s}[1] \cdot \mathfrak{a}[1] - \sum_{i=2}^{N} \mathfrak{s}[i] \cdot \mathfrak{a}[N - i + 2]$.

If we set $\mathbf{s} = \mathfrak{s}$ and $\mathbf{a}$ such that $\mathbf{a}[1] = \mathfrak{a}[1]$ and $\mathbf{a}[i] = -\mathfrak{a}[i]$ for $i = 2 \dots N$, then $(b, \mathbf{a})$ is a valid LWE sample with respect to $\mathbf{s}$.

Proof. (Key Switching, Lemma 6). Let us first note that for all $i \in [n]$ we have

$$\begin{aligned}\mathbf{x}^\top \cdot \mathsf{ksK} &= \sum_{i=1}^{N} \sum_{j=1}^{\ell_{\mathsf{ksK}}} \mathbf{x}[\ell_{\mathsf{ksK}}(i-1) + j] \cdot \mathsf{ksK}[\ell_{\mathsf{ksK}}(i-1) + j] \\ &= \mathsf{LWE}_{\sigma_1, n, Q}(\mathbf{s}, \sum_{i=1}^{N} \mathbf{a}[i] \cdot \mathbf{s}'[i])\end{aligned}$$

where

$$\begin{aligned}\sigma_1^2 &\leq \sum_{i=1}^{\ell_{\mathsf{ksK}}} N \cdot \mathsf{B}(\mathsf{G}^{-1}_{\mathsf{det}}(., \mathsf{L}_{\mathsf{ksK}})) \cdot \sigma^2_{\mathsf{ksK}} \\ &\leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{B}(\mathsf{G}^{-1}_{\mathsf{det}}(., \mathsf{L}_{\mathsf{ksK}})) \cdot \sigma^2_{\mathsf{ksK}}.\end{aligned}$$

$\mathsf{KeySwitchSetup}(\sigma_{\mathsf{ksK}}, \mathbf{s}, \mathbf{s}')$:

Input:
A bound $\sigma_{\mathsf{ksK}} \in \mathbb{N}$.
Secret keys $\mathbf{s} \in \mathbb{Z}_Q^n$, and $\mathbf{s}' \in \mathbb{Z}_Q^N$.

1 : For $i \in [N]$, $j \in [\ell_{\mathsf{ksK}}]$
2 : Set $\mathsf{ksK}[\ell_{\mathsf{ksK}}(i-1)+j] \leftarrow \mathsf{LWE}_{\sigma_{\mathsf{ksK}},n,Q}(\mathbf{s}, \mathbf{s}'[i] \cdot \mathsf{L}_{\mathsf{ksK}}^{j-1})$.
3 : Output $\mathsf{ksK} \in \mathsf{LWE}_{\sigma_{\mathsf{ksK}},n,Q}(\mathbf{s}', .)^{N\ell_{\mathsf{ksK}}}$.

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

Input: A LWE ciphertext $\mathbf{c} = [b, \mathbf{a}] \in \mathsf{LWE}_{\sigma,N,Q}(\mathbf{s}', m)$
A key switching key $\mathsf{ksK} \in \mathsf{LWE}_{\sigma_{\mathsf{ksK}},n,Q}(\mathbf{s}, .)^{N\ell_{\mathsf{ksK}}}$.

1 : Compute $\mathbf{x} \leftarrow \mathsf{G}_{\mathsf{det}}^{-1}(\mathbf{a}, \mathsf{L}_{\mathsf{ksK}}) \in \mathbb{Z}_{\mathsf{L}_{\mathsf{ksK}}}^{N\ell_{\mathsf{ksK}}}$.
2 : Output $c_{\mathsf{out}} \leftarrow [b, \mathbf{0}] - \mathbf{x}^\top \cdot \mathsf{ksK} \in \mathbb{Z}_Q^{n+1}$.

Fig. 4. Key switching algorithm and its setup.

The bound follows from the fact that we have a multisum of scalars in $\mathbf{x}$ and LWE samples from the key switching key.

Let us denote $b = \mathbf{a}^\top \cdot \mathbf{s}' + m + e$ and $\mathbf{x}^\top \cdot \mathsf{ksK} = [\hat{b}, \hat{\mathbf{a}}]$ where $\hat{b} = \hat{\mathbf{a}}^\top \mathbf{s} + \mathbf{a}^\top \cdot \mathbf{s}' + \hat{e}$ then

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

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

$$\sigma_{\mathsf{out}}^2 \leq N \cdot \ell_{\mathsf{ksK}} \cdot \mathsf{B}(\mathsf{G}_{\mathsf{det}}^{-1}(., \mathsf{L}_{\mathsf{ksK}})) \cdot \sigma_{\mathsf{ksK}}^2 + \sigma^2.$$

Proof. (Bootstrapping, Lemma 7). The correctness of blind rotation follows from two observations. First, is that multiplying a RLWE sample with X^k for some $k \in \mathbb{Z}_N$ does not change the parameter of its noise, because the error polynomial is only rotated, and we change the sign of some of the coefficients. Second, we run n times the homomorphic CMux gate, thus the variance of the output noise follows from Lemma 3. Finally, note that each iteration rotates the message by $X^{-\mathbf{a}[i] \cdot \mathbf{s}[i]}$. Denote $\mathbf{c} = [b, \mathbf{a}]$, where $b = \mathbf{a}^\top \mathbf{s} + m + e \in \mathbb{Z}_{2N}$. After n iterations we obtain $\mathfrak{a}_{\mathsf{rot}} \cdot X^{b - \mathbf{a}^\top \mathbf{s}} = \mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e}$. What follows is $(\mathfrak{a}_{\mathsf{rot}} \cdot X^{m+e})[1] = f(m+e) \in \mathbb{Z}_Q$ from the assumption on $\mathfrak{a}_{\mathsf{rot}}$.

$\mathsf{FunctionalBootstrap}^{\mathsf{ver}}(\mathsf{br}, \mathbf{u}, \mathsf{ksK}, \mathbf{c}, \mathfrak{a}_{\mathsf{rot}}, t)$:

Input:

A blind rotation key $\mathsf{br} = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathbf{s}, .)^n$.

A key switch key $\mathsf{ksK} \in \mathsf{LWE}_{\sigma_{\mathsf{ksK}},n,Q}(\mathbf{s}, .)^{\ell_{\mathsf{ksK}} N}$.

A LWE sample $\mathbf{c} = \mathsf{LWE}_{\sigma,n,q}(\mathbf{s}, .) = [b, \mathbf{a}] \in \mathbb{Z}_Q^{N+1}$.

A polynomial $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$.

An integer $t \in \mathbb{N}$.

1 : $\mathbf{c}_{\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}, \mathsf{ksK}) \in \mathbb{Z}_Q^{n+1}$.

2 : $\mathbf{c}_{\mathsf{pre}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{\mathsf{ksK}}, N) + \left[\lfloor \frac{N}{2t} \rceil, \mathbf{0}\right]$.

3 : $\mathsf{acc}_{\mathsf{sgn}} \leftarrow \mathsf{BlindRotate}^{\mathsf{det}}(\mathsf{br}, \mathfrak{a}_{\mathsf{sgn}}, \mathbf{c}_{\mathsf{pre}})$.

4 : $\mathbf{c}_{\mathsf{msb}} \leftarrow \mathsf{LWE\text{-}Ext}(\mathsf{acc}_{\mathsf{sgn}}, 1)$.

5 : $\mathbf{c}_{\mathsf{msb,ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}_{\mathsf{msb}}, \mathsf{ksK}) \in \mathbb{Z}_Q^{n+1}$.

6 : $\mathbf{c}_{\widehat{\mathsf{msb}}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{\mathsf{msb,ksK}}, 2N) \in \mathbb{Z}_{2N}^{n+1}$.

7 : $\mathbf{c}_{\mathsf{in}} \leftarrow \mathbf{c}_{\mathsf{pre}} + \mathbf{c}_{\widehat{\mathsf{msb}}} - \frac{2N}{4} \in \mathbb{Z}_{2N}^{n+1}$.

8 : $\mathsf{acc}_{\mathsf{out}} \leftarrow \mathsf{BlindRotate}^{\mathsf{ver}}(\mathsf{br}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{c}_{\mathsf{in}})$.

9 : Return $\mathbf{c}_{\mathsf{out}} \leftarrow \mathsf{LWE\text{-}Ext}(\mathsf{acc}_{\mathsf{out}}, 1)$.

Fig. 5. Bootstrapping: The full domain functional bootstrapping from [62, 52]. For the functional bootstrapping we additionally use a rotation polynomial $\mathfrak{a}_{\mathsf{sgn}}$ that is chosen such that the blind rotation computes a special $\mathsf{msb}(.)$ function of the input.

Correctness of bootstrapping trivially follows from the correctness of the underlying algorithms. In particular, the noise parameter of the $\mathbf{c}_{\mathsf{in}}$ ciphertext follows from the fact that we run the key switching procedure on $\mathbf{c}$ and then switch the modulus to $2N$. Finally, the noise parameter of $\mathbf{c}_{\mathsf{out}}$ follows from the correctness of blind rotation, Lemma 6, Lemma 5 and Lemma 4. Finally, if $\mathsf{ver} = \mathsf{simul}$, then we additionally compute a linear combination of of LWE samples of zero from the vector $\mathbf{v}$. Hence the additional part $\ell_R \cdot \sigma_R^2 \cdot \sigma_{\mathsf{rand}}^2$ of the noise follows from linear homomorphism of LWE samples and the fact that all error terms are uncorrelated.

Correctness of the full domain functional bootstrapping is as follows. Denote $\mathbf{c}_{\mathsf{pre}} = [b_{\mathsf{pre}}, \mathbf{a}_{\mathsf{pre}}]$ such that $b_{\mathsf{pre}} = \mathbf{a}_{\mathsf{pre}}^\top \cdot \mathbf{s} + m_{\mathsf{pre}} + e_{\mathsf{pre}} + \in \mathbb{Z}_N$. Note that since we add $\left[\lfloor \frac{N}{2t} \rceil, \mathbf{0}\right]$ we ensure that $0 \leq m_{\mathsf{pre}} + e_{\mathsf{pre}} < N$. Note that this shifting operation is important as otherwise we would not be able to choose an appropriate rotation polynomial. Assuming, that the phase of $\mathbf{c}_{\mathsf{pre}}$ is in $[0, N)$, we set all coefficients of the rotation polynomial $\mathfrak{a}_{\mathsf{sgn}}$ to $Q/4$. We blind rotate $\mathbf{c}_{\mathsf{pre}}$ modulo $2N$ with $\mathfrak{a}_{\mathsf{sgn}}$, so $b_{\mathsf{pre}} - \mathbf{a}_{\mathsf{pre}}^\top \cdot \mathbf{s} = m_{\mathsf{pre}} + e_{\mathsf{pre}} + kN \mod 2N$ for some $k \in \{0, 1\}$, where m_{pre} is the modulus switching of the message m that is encoded in $\mathbf{c}$. From correctness of blind rotation we have that, and then key and modulus switching we have that $\mathbf{c}_{\widehat{\mathsf{msb}}}$ decrypts to $\frac{2N}{4}$ if $k = 0$ and $\frac{3 \cdot 2N}{4}$ if $k = 1$. We can write the decryption of $\mathbf{c}_{\widehat{\mathsf{msb}}}$ as $k \cdot \frac{6N}{4} + (1-k) \cdot \frac{2N}{4}$. So when we add $\mathbf{c}_{\mathsf{pre}} + \mathbf{c}_{\widehat{\mathsf{msb}}} - \frac{2N}{4}$, the term $kN + k \cdot \frac{6N}{4} + (1-k) \cdot \frac{2N}{4} - \frac{2N}{4}$ is zero for both $k \in \{0, 1\}$. Hence we have $b_{\mathsf{in}} = \mathbf{a}^\top \cdot \mathbf{s} + m_{\mathsf{pre}} + e_{\mathsf{in}} \mod 2N$, where $m_{\mathsf{pre}} + e < N$. Therefore, we can can choose the coefficients of the rotation polynomial such that $\mathfrak{a}_{\mathsf{rot}} = F(\lfloor \frac{N}{t} m_{\mathsf{pre}} + e \rceil)$. Note that we will only multiply the rotation polynomials by $X^{m_{\mathsf{pre}}+e}$, where $0 \leq m_{\mathsf{pre}} + e < N$. In particular, the negacyclicity problem never occurs. In other words, we directly set the coefficient to encode the lookup table, and we do not worry that the rotation exceeds the number of coefficients and changes the sign of the output. Finally, the variance $\mathsf{Var}(e_{\mathsf{in}})$ follows from the error analysis of blind rotation, key switching, and modulus reduction. And σ_{out} follows from the analysis of blind rotation.

Numerical Error. Finally, let us address the issue of numerical errors when performing ring operations. In particular we focus on computing products of ring elements (or negacyclic convolutions of polynomials for our ring choise). Let $(\mathfrak{b}, \mathfrak{a})$ by a RLWE ciphertext such that $\mathfrak{b} - \mathfrak{a} \cdot \mathfrak{s} = \mathfrak{e}$, where $\mathfrak{e}$ is small. Denote

$$\begin{aligned}(\mathfrak{b}', \mathfrak{a}') &= (\mathbf{Mul}(\mathfrak{c}, \mathfrak{b}), \mathbf{Mul}(\mathfrak{a}, \mathfrak{c})) \\ &= (\mathfrak{b} \cdot \mathfrak{c} + \mathfrak{r}_1, \mathfrak{a} \cdot \mathfrak{c} + \mathfrak{r}_2),\end{aligned}$$

where $\mathfrak{r}_1$ and $\mathfrak{r}_2$ is the numerical error introduce by the multiplication algorithm $\mathbf{Mul}$. Then we can see that the phase $\mathfrak{b}' - \mathfrak{a}' \cdot \mathfrak{s} = \mathfrak{e} - \mathfrak{r}_2 \cdot \mathfrak{s} + \mathfrak{r}_1$. We obtain an additional error which infinity norm is

$$||\mathfrak{r}_2 \cdot \mathfrak{s} + \mathfrak{r}_1|| \leq NB(Q, \mathfrak{c}) \cdot (||s||_\infty + 1),$$

where $||\mathfrak{r}_1||_\infty, ||\mathfrak{r}_1||_\infty < B(Q, \mathfrak{c})$ and $B(Q, \mathfrak{c})$ being an error function determined by the modulus Q and the polynomial $\mathfrak{c}$. If the external product is implemented

using such errorenous multiplication algorithm then we need to add $2\ell_{\mathsf{br}} \cdot N \cdot B(Q,\mathfrak{c}) \cdot (||s||_\infty + 1)$ to the variance σ_1 assuming that the error function $B(Q,\mathfrak{c})$ is modeled by a discrete Gaussian. Consequently we need to update the bound on σ_{out} on the $\mathbf{c}_{\mathsf{out}}$ error of the bootstrapping algorithm as follows. For $\mathsf{ver} = \mathsf{det}$ we have

$$\sqrt{2n \cdot \ell_{\mathsf{br}} \cdot N \cdot (\sigma_{\mathsf{br}}^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{det}}^{-1}(.,\mathsf{L}_{\mathsf{br}})) + B(Q,\mathfrak{c}) \cdot (||s||_\infty + 1))},$$

and for $\mathsf{ver} = \mathsf{simul}$ we have

$$\sqrt{2n \cdot \ell_{\mathsf{br}} \cdot N \cdot (\sigma_{\mathsf{br}}^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{simul}}^{-1}(.,\mathsf{L}_{\mathsf{br}};\sigma_{\mathbf{x}})) + \cdot B(Q,\mathfrak{c}) \cdot (||s||_\infty + 1)) + \ell_R \cdot \sigma_R^2 \cdot \sigma_{\mathsf{rand}}^2}.$$

B Circuit Privacy for FHEW-style Blind Rotation

We recall the FHEW blind rotation algorithm [29] at Figure 6. Theorem 3 gives the distribution of the blind rotation when we plug the FHEW blind rotation algorithm into the bootstrapping procedure from Figure 3, instead of the TFHE blind rotation algorithm from Figure 2. We highlight the differences between Theorem 1 and Theorem 3 with a red box.

Theorem 3 (Distribution of the Bootstrap with FHEW-Style Blind Rotation). *Let* br *be the blind rotation key,* $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$ *a rotation polynomial, and* $\mathbf{c} \in \mathsf{LWE}_\sigma(\mathbf{s}, m)$ *a LWE sample as defined in the* Bootstrap *algorithm in Figure 2. Assume that* $\mathfrak{a}_{\mathsf{rot}}$ *is such that* $f(m) = (\mathfrak{a}_{\mathsf{rot}} \cdot X^{\mathsf{Phase}(\mathbf{c}_{\mathsf{in}})})[1]$ *where* $\mathbf{c}_{\mathsf{in}}$ *is the LWE sample obtained at step 2 of the* Bootstrap *algorithm. Let* $\mathbf{c}_{\mathsf{out}}$ *be the LWE sample returned by the* Bootstrap *algorithm for* $\mathsf{ver} = \mathsf{simul}$ *and Gaussian parameters* σ_{rand} *and* $\sigma_{\mathbf{x}}$ *where the Gaussian sampling algorithm* $\mathsf{G}_{\mathsf{simul}}^{-1}$ *is as in Lemma 1. Assume that*

$$\sigma_{\mathsf{rand}} \geq \max\left(4 \cdot \left((1-\gamma)(2\epsilon)^2\right)^{-1/\ell_R}, \sqrt{1+B_R} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}_R^2+1}\right) \cdot C_{\gamma,\ell_R}\right)$$

and

$$\sigma_{\mathbf{x}} \geq \sqrt{1+B_{\mathsf{br}}} \cdot \max\left(||\mathbf{q}_{\mathsf{L},Q}||, \sqrt{\mathsf{L}_{\mathsf{br}}^2+1}\right) \cdot \boxed{C_{\delta,2\cdot n\cdot \ell_{\mathsf{FHEW}}\cdot N\cdot \ell_{\mathsf{br}}}},$$

where B_{br} *and* B_R *are bounds on the infinity norm of the noise terms in the blind rotation key* br *and the masking vector* $\mathbf{v}$*. Then we have*

$$\Delta(\mathbf{c}_{\mathsf{out}}, \mathbf{c}_{\mathsf{fresh}}) \leq \max\left(\epsilon + 2\gamma, 2\delta\right),$$

where $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$, $b_{\mathsf{fresh}} = \langle \mathbf{a}_{\mathsf{fresh}}, \mathbf{s}' \rangle + f(m) + e_{\mathsf{rand}} + e_{\mathsf{out}}$, $e_{\mathsf{rand}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{rand}}\cdot\sqrt{1+\ell_R\sigma_R^2}}$, *and* $\boxed{e_{\mathsf{out}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}\sqrt{1+2n\ell_{\mathsf{FHEW}}N\sigma_{\mathsf{br}}}}}$.

Proof (Sketch). The proof is nearly the same as the proof for Theorem 1. The difference is that the FHEW blind rotation consists of a sequence of external products, whereas the TFHE algorithm consists of a sequence of MUX gates.

$\mathsf{BRKeyGen}(\sigma_{\mathsf{br}}, \mathsf{L}_{\mathsf{FHEW}}, \mathfrak{s}, \mathbf{s})$:

Input:

An error distribution σ_{br}.

A decomposition base $\mathsf{L}_{\mathsf{FHEW}}$, where $\ell_{\mathsf{FHEW}} = \lceil \log_{\mathsf{L}_{\mathsf{FHEW}}} 2 \cdot N \rceil$.

A RLWE secret key $\mathfrak{s} \in \mathcal{R}_Q$.

A LWE secret key $\mathbf{s} \in \mathbb{Z}_t^n$.

1 : For $i \in [n]$, $v \in [0, \mathsf{L}_{\mathsf{FHEW}} - 1]$ and $j \in [\ell_{\mathsf{FHEW}}]$

2 : Set $\mathsf{br}[i, j, v] = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, X^{v \cdot \mathbf{s}[i] \cdot \mathsf{L}_{\mathsf{FHEW}}^{j-1}})$.

3 : Output $\mathsf{br} \in \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, .)^{n \times \ell_{\mathsf{FHEW}} \times \mathsf{L}_{\mathsf{FHEW}}}$.

$\mathsf{BlindRotate}^{\mathsf{ver}}(\mathsf{br}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{c}; \sigma_{\mathbf{x}})$:

Input:

A blind rotation key $\mathsf{br} = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s}, .)^{n \times \ell_{\mathsf{FHEW}} \times \mathsf{L}_{\mathsf{FHEW}}}$.

An rotation polynomial $\mathfrak{a}_{\mathsf{rot}} \in \mathcal{R}_Q$.

A ciphertext $\mathbf{c} \in \mathsf{LWE}_{\sigma,n,2N}(\mathbf{s}, .)$.

[If simul] A Gaussian param. $\sigma_{\mathbf{x}}$.

1 : Let $\mathbf{c} = [b, \mathbf{a}] \in \mathbb{Z}_{2N}^{n+1}$.

2 : Set $\mathbf{c}_{\mathsf{acc},0} \leftarrow [\mathfrak{a}_{\mathsf{rot}} \cdot X^b, 0] \in R_Q^2$

3 : Let $\mathbf{V} \leftarrow \mathsf{G}^{-1}(\mathbf{s}) \in \mathbb{Z}_{\mathsf{L}_{\mathsf{FHEW}}}^{n \times \ell_{\mathsf{FHEW}}}$.

4 : For $i \in [n]$:

5 : For $j \in [\ell_{\mathsf{FHEW}}]$:

6 : $\mathbf{c}_{\mathsf{acc},\ell_{\mathsf{FHEW}} \cdot (i-1)+j-1} \leftarrow \mathsf{extProd}(\mathsf{br}[i, j, \mathbf{V}[i, j]], \mathbf{c}_{\mathsf{acc},\ell_{\mathsf{FHEW}} \cdot (i-1)+j-2}; \sigma_{\mathbf{x}})$.

7 : Output $\mathbf{c}_{\mathsf{acc},n \cdot \ell_{\mathsf{FHEW}}} \in R_Q^2$.

Fig. 6. FHEW-style Blind Rotation and its Setup.

Thereby, for TFHE, an important part of the proof is to show that after the sequence of MUX gates, the error term is in the form as given by Equation 2. For the FHEW algorithm, we have that the error is already in the required form, which follows from Equation 1 in Section 4. In particular, we have

$$\mathsf{Error}(\mathbf{c}_{\mathsf{acc}}) = \mathsf{Error}(\mathbf{c}_{\mathsf{acc},\ell_{\mathsf{FHEW}}\cdot n-1}) = \sum_{i=1}^{n}\sum_{j=1}^{\ell_{\mathsf{FHEW}}} \widehat{\mathfrak{e}}_{i,j}\cdot X^{-\sum_{k=i}^{n}\sum_{l=j}^{\ell_{\mathsf{FHEW}}}\mathbf{V}[k,l]\cdot\mathbf{s}[k]\cdot\mathsf{L}_{\mathsf{FHEW}}^{l-1}}.$$

To summarize, the error term after FHEW blind rotation and extraction is

$$\mathsf{Error}(\mathbf{c}_{\mathsf{ext}}) = \sum_{i=1}^{n}\sum_{j=1}^{\ell_{\mathsf{FHEW}}}\sum_{k=1}^{2\ell_{\mathsf{FHEW}}}\sum_{l=1}^{N}\mathbf{e}_{i,j,k}[l]\cdot\mathfrak{x}_{i,j,k}[l].$$

The difference with TFHE is the number of external products. The rest of the proof follows the same hybrids as in the proof of Theorem 1.

C Additional Preliminaries

In this section we recall some useful lemmas.

Lemma 12 (Smudging Lemma [9]). *Let B_1 and B_2 be two be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer. Let $e_2 \leftarrow_{\$} [-B_2, B_2]$ be chosen uniformly at random. Then the statistical distance between e_2 and $e_2 + e_1$ is*

$$\Delta(e_2, e_2 + e_1) = B_1/B_2.$$

Lemma 13 (Lemma 2.3 from [30]). *Let $\delta \in [0,1]$ and $f : \mathcal{S} \to \mathcal{S}$ be a randomized function such that $\Delta(f(a), f(b)) \leq \delta$ holds for all $a, b \in \mathcal{S}$. Then*

$$\forall k \leq 0, \forall a, b \in \mathcal{S}, \quad \Delta(f^k(a), f^k(b)) \leq \delta^k,$$

where f^k denotes composing the function f k-times.

C.1 Gaussian Sampling Algorithms

We recall the Gaussian sampling algorithm form [32] on Figure 8. For completeness, we also recall the Gaussian sampling algorithm from [56] on Figure 7 that is specialized for modulus in the form $Q = \mathsf{L}^{\ell}$. We recall only the specification for sampling given a one-dimensional integer target. Remind that we use the sampling algorithm separately on every coefficient when given as input a polynomial.

Note that for the special case modulus, the standard deviation can be bounded by $\mathsf{L} \cdot C_{\epsilon,\ell}$ which is much smaller than for the general case of $Q < \mathsf{L}^{\ell}$. Our correctness estimation scripts take all the decomposition algorithms into account. In practice, however, a modulus of the form $Q = \mathsf{L}^{\ell}$ forces us to implement

$\mathsf{G}^{-1}_{\mathsf{simul}}(a, \mathsf{L}, \sigma_{\mathbf{x}})$:

Input:

Integers $a \in \mathbb{Z}_Q$ and L s.t. $Q = \mathsf{L}^\ell$ where $\ell \in \mathbb{N}$.
A Gaussian parameter $\sigma_{\mathbf{x}}$.

1 : For each $j \in [0, \ell - 1]$:
2 : $\quad \mathbf{x}[j] = \mathcal{D}_{\mathsf{L}\mathbb{Z}+a,\sigma_{\mathbf{x}}}$.
3 : $\quad a \leftarrow (a - \mathbf{x}[i][j])/\mathsf{L}$.
4 : Output $\mathbf{x}$.

Fig. 7. Gaussian Sampling Algorithm [56].

negacyclic convolution of polynomials with fast Fourier transforms on floating point arithmetic. As discussed in Section 5, we found it infeasible to instantiate the scheme such that no numerical errors are induced by ring multiplications.

The algorithm on Figure 8 takes additionally the following precomputed vectors as input. The vector $\mathbf{l}$ is such that $\mathbf{l}[1]^2 = \mathsf{L}(1 + 1/\ell)$ and $\mathbf{l}[i]^2 = \mathsf{L}(1 + 1/(\ell - 1))$. The vector $\mathbf{h}$ is such that $\mathbf{h}[1] = 0$ and $\mathbf{h}[i + 1]^2 = \mathsf{L}(1 + 1/(\ell - 1))$ for $i \in [2, \ell]$. Finally, we assume that any vector at index 0 and $\ell + 1$ is set to zero. For more details on the correctness of the Gaussian sampling algorithm for any $Q < \mathsf{L}^\ell$ we refer to [32]. furthermore, we refer to [56] for the analysis of the Gaussian sampling algorithm on Figure 7 for the special case $Q = \mathsf{L}^\ell$.

$\mathsf{G}^{-1}_{\mathsf{simul}}(a, \mathsf{L}, \sigma_{\mathbf{x}})$:

Input:

Integers $a \in R_Q$ and L s.t. $Q = \mathsf{L}^\ell$, and a Gaussian parameter $\sigma_{\mathbf{x}}$.

Set $\mathbf{q} \leftarrow \mathsf{G}^{-1}_{\mathsf{det}}(Q, \mathsf{L})$ and $\mathbf{u} \leftarrow \mathsf{G}^{-1}_{\mathsf{det}}(a, \mathsf{L})$.
$\sigma = \sigma_{\mathbf{x}}/(\mathsf{L}+1)$.
$\mathbf{p} \leftarrow \mathsf{Perturb}(\sigma, \sigma_{\mathbf{x}})$.
For each $i \in [\ell]$
  $\mathbf{c}[i] \leftarrow (\mathbf{c}[i-1] - \mathbf{u}[i] - \mathbf{p}[i])/\mathsf{L}$.
$\mathbf{z} \leftarrow \mathsf{SampleD}(\sigma, \mathbf{c}, \sigma_{\mathbf{x}})$.
For each $i \in [\ell - 1]$
  $\mathbf{t}[i] \leftarrow \mathsf{L} \cdot \mathbf{z}[i] - \mathbf{z}[i-1] + \mathbf{q}[i] \cdot \mathbf{z}[\ell - i] + \mathbf{u}[i]$.
$\mathbf{t}[\ell] \leftarrow \mathbf{q}[\ell] \cdot \mathbf{z}[\ell] - \mathbf{z}[\ell - 1] + \mathbf{u}[\ell]$.
Return $\mathbf{t}$.

$\mathsf{SampleD}(\sigma, \mathbf{c}, \sigma_{\mathbf{x}})$:

Input:

Gaussian parameters σ and $\sigma_{\mathbf{x}}$, and a vector $\mathbf{c} \in \mathbb{R}^\ell$.

$\mathbf{z}[\ell] \leftarrow \lfloor -\mathbf{c}[\ell]/\mathbf{d}[\ell] \rfloor$.
$\mathbf{z}[\ell] \leftarrow \mathbf{z}[\ell] + \mathsf{SampleZ}_t(\sigma/\mathbf{d}[\ell], \lfloor -\mathbf{c}[\ell]/\mathbf{d}[\ell] \rfloor_{[0,1)}, \sigma_{\mathbf{x}})$.
$\mathbf{c} \leftarrow \mathbf{c} - \mathbf{z}[\ell] \cdot \mathbf{d}$.
For all $i \in [\ell - 1]$.
  $\mathbf{z}[i] \leftarrow \lfloor -\mathbf{c}[i] \rfloor + \mathsf{SampleZ}_t(\sigma, \lfloor -\mathbf{c}[i] \rceil_{[0,1)}, \sigma_{\mathbf{x}})$.
Return $\mathbf{z}$.

$\mathsf{Perturb}(\sigma, \sigma_{\mathbf{x}})$:

Input:

A Gaussian parameters σ and $\sigma_{\mathbf{x}}$.

$\beta \leftarrow 0$.
For i **in** $[\ell]$:
  $\mathbf{c} \leftarrow \beta/\mathbf{l}[i]$ and $\sigma[i] \leftarrow \sigma/\mathbf{l}[i]$
  $\mathbf{z} \leftarrow \lfloor \mathbf{c}[i] \rfloor + \mathsf{SampleZ}_t(\sigma[i], \lfloor \mathbf{c}[i] \rceil_{[0,1)}, \sigma_{\mathbf{x}})$.
  $\beta \leftarrow -\mathbf{z}[i]\mathbf{h}[i]$.
$\mathbf{p}[1] \leftarrow (2\mathsf{L}+1)\mathbf{z}[1] + \mathsf{L}\mathbf{z}[1]$.
For i **in** $[2, \ell]$:
  $\mathbf{p} \leftarrow \mathsf{L}(\mathbf{z}[i-1] + 2\mathbf{z}[i] + [i+1])$.
Return $\mathbf{p}$.

Fig. 8. Gaussian Sampling Algorithm [32] for $Q < \mathsf{L}^\ell$. We denote $\lfloor c \rceil_{[0,1)} = c - \lfloor c \rfloor$. The algorithm $\mathsf{SampleZ}_t(\sigma, c, \sigma_{max})$ is any Gaussian sampling algorithm that samples over $\mathbb{Z} \cup [c - t \cdot \sigma_{max}, c + t \cdot \sigma_{max}]$ with mean c. We assume