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

Kamil Kluczniak

secunet Security Networks AG[**]
kamil.kluczniak@gmail.com

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 the 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 between 1.3 and 0.9 seconds, depending which Gaussian sampling algorithm is used, and whether the parameter set targets a fast Fourier or a number theoretic transform-based implementation. In addition, we can perform non-sanitized bootstrapping in around 0.27 or 0.14 seconds. Crucially, we do not need to increase the parameters to perform computation before or after 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

[**] The work was mostly done while I was at CISPA Helmholtz Center for Information Security.

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 [RAD78], and the first theoretical realization of that concept is due to Gentry [Gen09b].

A critical property for FHE is circuit privacy (also called function 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 [Nui22]. 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 [HFH99, Mea86, CLR17], oblivious pseudorandom functions [BIP$^+$18, ADDG23] neural network inference [DGBL$^+$16, CdWM$^+$17, LJLA17, JKLS18, JVC18, BGGJ18, ABSdV19, CDKS19, RSC$^+$19, BGPG20] or analysis on genomic data [KSK$^+$18, KSK$^+$20, BGPG20]. Recently, Akavia, Gentry, Halevi, and Vald [AGHV22] showed that circuit private IND-CPA secure homomorphic encryption satisfies a relaxed notion of CCA2 security. 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, as current FHE schemes require sending public keys and ciphertexts that are an 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 [GH11, BV11, BGV12, GHS12, AP13, GSW13, BV14, AP14, HS15, DM15, CGGI16a, CH18, CGGI20, HS21], and numerous implementations [PAL21, CGGI16b, CJL$^+$20, Lat22] 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 [Gen09b]. 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 may depend 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: The technique 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 requires us to choose very big parameters. We note that it is required to take the noise super-polynomially larger than the noise in the sanitized 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 [CLR17] where we do not bootstrap the ciphertexts and can tolerate larger parameters.

Ducas-Stehlé Washing Machine: Introduced by Ducas and Stehlé [DS16], 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 [DM15] 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 [GHV10, CO17] proposed to use garbled circuit-based techniques to provide circuit privacy. For example Gentry, Halevi, Vaikuntanathan [GHV10] 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 [CO17] propose to use garbled circuits to compute the decryption step.

Rerandomizing Computation: Bourse, Pino, Minelli and Wee [BDPMW16]. exploits properties of the Gentry, Sahai, Waters (GSW) cryptosystem [GSW13] to build a circuit private homomorphic encryption scheme. Specifically, when multiplying GSW ciphertexts, they use a randomized version of gadget decomposition instead of a deterministic one. In [BDPMW16] the authors show that when gadget decomposition is implemented via Gaussian sampling [MP12, GM18] with appropriate parameters, then we can build an FHE scheme for circuits in NC^1 (circuit of depth logarithmic in the number of inputs). The results are asymptotic, without concrete parameter proposals nor implementation.

1.1 Our Contributions.

We design a randomized FHEW/TFHE-style [DM15, CGGI16a] bootstrapping algorithm that can sanitize a given ciphertext. In contrast to the Ducas-Stehlé washing machine method [DS16], 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 [BDPMW16], in that we use their randomization concept in an FHEW-style bootstrapping scheme. We note that porting the ideas from [BDPMW16] to the ring setting is non-trivial since there are no analogues of the leftover hash lemma [DRS04] and Gaussian leftover hash lemma [AGHS13, AR13, BDPMW16] for the ring setting. Moreover, we can argue that designing a "leaky" analogue of the regularity lemmas [SS11, LPR13] as in [DSGKS21], may result in practically inefficient bootstrapping. While we use some techniques from [BDPMW16], the overall method departs from [BDPMW16]. In this work we show how to bypass the need to port [BDPMW16] into the ring setting using simple techniques in the right places by exploiting structural properties of FHEW/TFHE [DM15, CGGI16a] 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 [BDPMW16] 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 [DS16] gives a heuristic instantiation based on FHEW [DM15], 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 [DS16] 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 [DM15, CGGI16a, CGGI20] 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 sanitizes a ciphertext in about 1.3 seconds for both the NTT and FFT-based implementations when using Karney's Gaussian sampling algorithm [Kar16]. When sampling from the rounded continuous Gaussian distribution via Box-Muller transform [BM58] we can reduce the times to

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

1.0 and 0.9 seconds for NTT and FFT, 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 is between 1.56× to 7.88× slower than ours, depending on the parameter sets and the implementation of the Gaussian sampling algorithm. We compared parameters with the same key sizes, non-sanitized (deterministic) computation time, and correctness level.

We also stress that Gaussian sampling in our method constitutes about 78% and 40% of the entire computation. Our implementation is linear and doesn't take advantage of special vector instructions or parallelism. Based on the speedups when using different Gaussian samplers, we believe that an optimized implementation could significantly improve the execution times, while we do not see much room for improvement in DS-WM as it simply repeats the base bootstrapping algorithm.

1.2 Our Techniques.

In this section, we first give a high-level overview of FHEW/TFHE-style bootstrapping and our circuit private version. Then we discuss the technical problems to realize the idea and our solutions.

FHEW/TFHE-Style Bootstrapping. First, let us recall the symmetric key version of Regev's encryption [Reg09]. The encryption algorithm chooses a vector $\mathbf{a} \in \mathbb{Z}_q^n$ and a secret key $\mathbf{s} \in \mathbb{Z}_q^n$, and computes an encryption of a message $m \in \mathbb{Z}_t$ as $[b, \mathbf{a}] \in \mathbb{Z}_q^{n+1}$, where $b = \langle \mathbf{a}, \mathbf{s} \rangle + \tilde{m} + e \pmod q$, $\tilde{m} = \frac{q}{t} \cdot m$, and $e < \frac{q}{2 \cdot t}$. For simplicity, we assume that $t \mid q$. The decryption algorithm calculates $\left\lfloor \frac{t}{q} \cdot \big(b - \langle \mathbf{a}, \mathbf{s} \rangle\big) \right\rceil = \left\lfloor \frac{t}{q} \cdot (\frac{q}{t} \cdot m + e) \right\rceil = m$.

The scheme can also be instantiated over the cyclotomic ring $\mathcal{R}_Q$, where $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. To encrypt a message $\mathfrak{m}$ from the ring $\mathcal{R}_t$, the scheme selects $\mathfrak{a} \in \mathcal{R}_Q$ and a secret key $\mathfrak{s} \in \mathcal{R}_Q$ and computes the encrypted message $[\mathfrak{b}, \mathfrak{a}]$ where $\mathfrak{b} = \mathfrak{a} \cdot \mathfrak{s} + \frac{q}{t} \cdot \mathfrak{m} + \mathfrak{e}$, and $\mathfrak{e}$ is a small error term in $\mathcal{R}_Q$.

Now let us proceed to the ideas underlying the FHEW-style bootstrapping scheme introduced by Ducas and Micciancio in [DM15]. We want to re-encrypt an LWE ciphertext $[b, \mathbf{a}] \in \mathbb{Z}_q^{n+1}$. Assuming that the LWE modulus is $q = 2 \cdot N$, the scheme sets up a homomorphic accumulator acc as an RLWE encryption of $\mathfrak{a}_{\mathsf{rot}} \cdot X^b \in \mathcal{R}_Q$. We refer to [DM15] on how to choose $\mathfrak{a}_{\mathsf{rot}}$.

Next, the scheme multiplies acc with encryptions of $X^{-\mathbf{a}[i] \cdot \mathbf{s}[i]} \in \mathcal{R}_Q$ for each $i \in [1, n]$. Finally, the message in the accumulator will be:

$$\mathfrak{a}_{\mathsf{rot}} \cdot X^{b - \sum_{i=1}^n \mathbf{a}[i] \cdot \mathbf{s}[i]} = \mathfrak{a}_{\mathsf{rot}} \cdot X^{k \cdot q + \tilde{m} + e} \qquad = \mathfrak{a}_{\mathsf{rot}} \cdot X^{\tilde{m} + e \bmod 2 \cdot N} \in \mathcal{R}_Q.$$

The last step is to extract an LWE encryption of the constant term from the rotated accumulator. Denote this LWE ciphertext as $[\mathbf{a}_{\mathsf{out}}, b_{\mathsf{out}}]$, where $b = \langle \mathbf{a}_{\mathsf{out}}, \mathbf{s} \rangle + m_{\mathsf{out}} + e_{\mathsf{out}}$.

How to Get Circuit Privacy. Note that the output ciphertext is entirely determined by the input ciphertext and the evaluation key. One solution is to use the noise flooding technique [Gen09a]. Essentially, the technique involves adding a fresh ciphertext and some uniform in an interval noise to the output ciphertext. Although this makes the noise in the output ciphertext uniform and independent, the magnitude of the noise term is exponential in the security parameter. This means that a large modulus (over 110 bits) must be chosen and the dimension must be increased to compensate for the security loss, resulting in a large evaluation key and reduced efficiency. We give sample estimates for this technique in Supplementary Material E.

Another solution by Ducas and Stehlé [DS16] is to apply smaller flooding noise and repeat the bootstrapping process $O(\lambda)$ times to achieve the desired level of circuit privacy security. However, this approach has the immediate downside of having to repeat the expensive bootstrapping operation multiple times. Nevertheless, in this paper we show new parameters and optimizations of [DS16] applied to FHEW/TFHE-style bootstrapping, in order to give better insights into the state of the art and a proper comparison with our method.

Our idea is to re-randomize the blind rotation algorithm so that the distribution of the extracted LWE ciphertext will already be independent of the input ciphertext. To do this, we first need to construct a randomized algorithm to multiply RLWE ciphertexts. Our starting point is the algorithm introduced by Bourse et al. in [BDPMW16], who showed a randomized product of GSW [GSW13] ciphertexts. It turns out, however, that there are multiple problems that we need to overcome to apply this high level idea for the bootstrapping algorithm.

Brief Overview of the Randomized GSW Product. 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. 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.

Now we are ready to recall the method from [BDPMW16]. 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 [BDPMW16] 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 [DRS04]. 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, [BDPMW16] use an adaptation of the Gaussian leftover hash lemma [AGHS13, AR13].

In FHEW/TFHE-style bootstrapping, we need to perform external products of ring LWE ciphertexts. Hence the immediate problem is to translate the randomization technique into the ring setting. But as we discuss later, such translation may still be impractical.

Problems with Translating [BDPMW16] Into the Ring Setting. At the heart of FHEW/TFHE-style bootstrapping algorithms is an algorithm called blind rotation that outputs an RLWE ciphertext $\mathbf{c} = (\mathfrak{a}, \mathfrak{b}) \in \mathcal{R}_Q^2$ 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. Recall that $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

If we want to follow the technique from [BDPMW16], 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 analogue 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 [LPR13] as in [DSGKS21]. But we argue that even if we would ignore the leak, the regularity lemma from [LPR13] 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 analogue of the Gaussian leftover hash lemma.

Our Solution. To bypass these problems, we exploit that in FHEW/TFHE-style schemes, we extract an LWE ciphertext from $\mathbf{c}$. In particular, observe that $(\mathbf{a}', b') \in \mathbb{Z}_Q^{N+1}$, 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 $i = 0 \ldots N-2$, 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 $(\mathbf{a}', b')$ obtaining a ciphertext $(\bar{\mathbf{a}}, \bar{b})$ of the same message where $\bar{\mathbf{a}}$ is statistically close to uniform. Finally, we can show that the error term of $(\mathbf{a}', b')$ is already in the form required by the Gaussian leftover hash lemma [AGHS13, AR13, BDPMW16]. 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. To summarize, our construction completely bypasses the need to adapt [BDPMW16] to the ring setting.

Concurrent Work. Concurrently and independently, Bourse and Izabachéne [BI22] gave a circuit private FHEW/TFHE-style algorithm. However, the techniques and practical efficiency differ significantly. Roughly speaking, [BI22] build a multiplication algorithm between RLWE and RGSW ciphertexts that outputs an RLWE ciphertext of the product, which is statistically close to a "fresh" ciphertext. To randomize the bootstrapping algorithm, [BI22] needs to publish a sanitization key that consists of $\geq 2^{17}$ RLWE ciphertexts alongside the bootstrapping key. Such key requires over 677 MB memory[2]. In contrast, our algorithm requires less than 186 or 69 MB (depending on the parameter set) of additional sanitization key. Furthermore, [BI22] needs to sample a "fresh" RLWE ciphertext for every external product in the blind rotation step. As reported in [BI22] sampling a single RLWE ciphertext takes approximately 45 seconds, and they need to sample 612 to perform a single bootstrapping operation. That gives us 7.6 hours to perform the sampling. To overcome the timing issue, the authors assume that the RLWE ciphertexts and Gaussian are precomputed, and the machine has unrestricted memory and precomputation. However, to make the computation feasible on a laptop they test the algorithm by reusing a single RLWE ciphertext. Even with the precomputed values, deterministic bootstrapping takes 3.15 seconds, and sanitization bootstrapping takes between 21 and 4.68 seconds depending on how many Gaussian samples were precomputed. In comparison, our deterministic bootstrapping takes 0.14 or 0.27 seconds, sanitization takes between 0.9 and 1.3 seconds. Most importantly, we do not need any precomputation for our algorithms to be efficient, and all random variables are generated on-the-fly.

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 $[n, \ldots, m]$. 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

[2] Assuming each integer is stored in a byte array. If integers are stored in 64-bit registers, then the key size grows to ≈ 0.9 GB of memory.

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

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 [Reg05]. Our description is a generalized version due to Brakerski, Gentry, and Vaikuntanathan [BGV12].

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. We define a Generalized Learning With Errors (GLWE) sample of a message $\mathfrak{m} \in \mathcal{R}_Q$ with respect to a secret key $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^n$, as*

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

where $\mathbf{a} \leftarrow_{\$} \mathcal{R}_Q^n$ *and* $\mathfrak{e} \leftarrow_{\$} \mathcal{D}_{\mathcal{R},\sigma}$. *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 [RAD78, Gen09b].

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* $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Eval}(\mathsf{ek},[\mathsf{ct}_i]_{i=1}^n,\mathcal{C})$.

Efficiency: *We require that* Setup, Enc *and* Dec *run in* $\mathsf{poly}(\lambda)$ *time, and* Eval *runs in* $\mathsf{poly}(\lambda,|\mathcal{C}|)$ *time. Finally, we say that fully homomorphic encryption is compact if the size of the output of* Eval *is independent of* $\mathcal{C}$. *Namely, if* $|\mathsf{Eval}(\mathsf{ek},[\mathsf{ct}_i]_{i=1}^n,\mathcal{C})|$ *is* $\mathsf{poly}(\lambda,|\mathcal{M}|)$.

Indistinguishability Under Chosen Plaintext Attack is defined in the usually way. For completeness we recall the formal definition in Supplementary material A.

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 if for fixed* $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\lambda)$ *and all* $c_1, \ldots, c_n$ *such that* $[m_i \leftarrow \mathsf{Dec}(\mathsf{sk}, c_i)]_{i=1}^n$ *and* $m_{\mathsf{out}} \leftarrow \mathcal{C}(m_1, \ldots, m_n)$ *there exists a* **PPT** *simulator* Sim *such that*

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

Our simulation-based definition of circuit privacy is stronger than [IP07, BDPMW16] in two aspects. First, our simulator does not require us to know the size of the circuit as in [IP07]. In fact, our simulator only needs to know the outcome of the circuit and nothing else. Second, we only assume that the ciphertexts input to the evaluator decrypt to the messages input to the circuit.

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 C 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 C.

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 $\mathbf{x} = \mathsf{G}_{\mathsf{ver}}^{-1}(\mathfrak{c}, \mathsf{L}; \sigma) \in \mathcal{R}^\ell$ 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 $\mathfrak{c} = \mathbf{g}_{\mathsf{L},Q}^\top \cdot \mathbf{x} \in \mathcal{R}_Q$. Note that $\mathsf{G}_{\mathsf{ver}}^{-1}$ also takes the modulus Q implicitly as input. A special case of the above is when $\mathsf{G}_{\mathsf{ver}}^{-1}$ 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 [MP12, GM18]. One for a general $Q < \mathsf{L}^\ell$ and one specialized for $Q = \mathsf{L}^\ell$. We recall both in Supplementary Material A, 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 $\mathsf{ver} = \mathsf{det}$, 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 [GMP19]. 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 [GM18]). *There exists a sampling 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 $\sigma_{\mathbf{x}}$, outputs $\mathbf{y} \in \mathbb{Z}^\ell$ such that*

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

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 [GSW13] on Figure 1. We also recall the external product [CGGI16a, CGGI20], 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.

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

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

Fig. 1. RGSW Encryption and External Product.

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* $\mathbf{c}_{\mathsf{out}} \in \mathsf{RLWE}_{\sigma_{\mathsf{out}}}(\mathbf{s}, \mathfrak{m}_{\mathsf{out}})$, *where* $\mathfrak{m}_{\mathsf{out}} = \mathfrak{m} \cdot \mathfrak{m}_G$ *and*

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

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 an RLWE encoding one of the message from $\mathbf{d}$ or $\mathbf{h}$ depending on the bit encoded in $\mathbf{C}$.

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* $m_{\mathbf{C}} \in \{0, 1\}$ *and* $\mathfrak{m}_{\mathbf{d}}, \mathfrak{m}_{\mathbf{h}} \in \mathcal{R}_Q$. *Optionally it also takes a Gaussian parameter* $\sigma_{\mathbf{x}}$. *In particular, the gate computes and outputs* $\mathsf{extProd}_{\mathsf{ver}}(\mathbf{C}_G, \mathbf{d} - \mathbf{h}; \sigma_{\mathbf{x}}) + \mathbf{h}$. *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* $\mathfrak{m}_{\mathsf{out}} = \mathfrak{m}_{\mathbf{h}}$ *for* $\mathsf{m}_{\mathbf{C}} = 0$ *and* $\mathsf{m}_{\mathsf{out}} = \mathfrak{m}_{\mathbf{d}}$ *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 [BV11] allows us to change the modulus of a given ciphertext without the knowledge of the secret key.

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, allows extracting an LWE 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* $\mathbf{c}_{\mathsf{out}} = [\mathbf{a}, b] \in \mathbb{Z}_Q^{N+1}$ *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. 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 C.

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 C we recall a version of the algorithm that applies a trick from [YXS+21, LMP21], 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 [KS22, YXS+21, CLOT21, LMP22, Klu22], 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*

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

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*

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

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.

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

Input:
An error distribution $\sigma_{\mathsf{br}}$.
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 $\mathbf{c} = [\mathbf{a}, b] \in \mathbb{Z}_{2N}^{n+1}$.
2 : Set $\mathbf{c}_{\mathsf{acc},0} \leftarrow [0, \mathfrak{a}_{\mathsf{rot}} \cdot X^b] \in R_Q^2$
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.

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[3]. Choose the radices L_{br}, $\mathsf{L}_{\mathsf{ksK}} \in \mathbb{N}$ 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 $\mathbf{v} \leftarrow \mathsf{LWE}_{\sigma_R,N,Q}(\mathbf{s}', 0)^h$. 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.

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. 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.

[3] Other distributions for the LWE secret key are possible. See [MP21] for an excelent summary.

```
$\mathsf{Bootstrap}_{\mathsf{ver}}(\mathsf{br}, \mathsf{ksK}, \mathbf{c}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{v}, \sigma_{\mathsf{rand}}, \sigma_{\mathbf{x}})$:
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 $\mathbf{v} = \mathsf{LWE}_{\sigma_R,N,Q}(\mathbf{s}', 0)^h$.
[If simul] Gaussian parameters $\sigma_{\mathsf{rand}}, \sigma_{\mathbf{x}}$.
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 $\mathbf{r} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}^h, \sigma_{\mathsf{rand}}}$ and $y \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}, \sigma_{\mathbf{x}}}$.
7 :     Set $\mathbf{c}_{\mathsf{rand}} \leftarrow \mathbf{v}^\top \cdot \mathbf{r}$.
8 :     Set $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{c}_{\mathsf{ext}} + \mathbf{c}_{\mathsf{rand}} + y$.
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.

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

4 Analysis of Circuit Privacy

This section contains our core analytical contribution. First, in Section 4.1, we state a few technical lemmas needed for the circuit-privacy analysis in Section 4.2.

4.1 Generalized (Gaussian) Leftover Hash Lemma.

Below we give our fixed and generalized version of the Gaussian Leftover hash lemma from [BDPMW16].

Lemma 8 (Gaussian Leftover Hash Lemma (Generalized Lemma 3.6 from [BDPMW16])). *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 [GPV08]), and a lemma (Lemma 3.7 in [BDPMW16]) 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 [BDPMW16], the authors prove the lemma for a modulus Q that is a power of two. We generalize the lemma to modulus of form $Q \leq \mathsf{L}^\ell$. For completeness, we recall the proof in Supplementary Material B and the necessary background in Supplementary Material A. Below we state our generalized version of Lemma 3.7 from [BDPMW16].

Lemma 9 (Generalized Lemma 3.7 from [BDPMW16]). *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*

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

We defer the additional background to Supplementary Material A.

Proof. We use Lemma 15 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:

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

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 $\mathbf{u}_1, \ldots, \mathbf{u}_m$ are linearly independent since $\mathsf{span}(\mathbf{u}_1, \ldots, \mathbf{u}_m, \mathbf{b}_{m+1}) = \mathsf{span}(\mathbf{b}_1, \ldots, \mathbf{b}_m, \mathbf{b}_{m+1}) = \mathbb{R}^{m+1}$ (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 give our version of the leftover hash lemma, which is an instantiation of the lemma from [DRS04, DORS08].

Lemma 10 (Leftover Hash Lemma). *Let $\epsilon > 0$ and Q be a odd prime. For any $\mathbf{e} \in \mathbb{Z}_Q^m$ and $\mathbf{C} \in \mathcal{R}^m$, if $\sigma_{\mathsf{rand}} \geq C_{\epsilon,m}$ then*

$$\Delta\big((\mathbf{A}\mathbf{r}, \mathbf{A}, \mathbf{e}^\top\mathbf{r}), (\mathbf{u}, \mathbf{A}, \mathbf{e}^\top\mathbf{r})\big) \leq \frac{1}{2}\sqrt{\frac{2^{(n+1)\log(Q)}}{2^{\log(1-\epsilon)+m\log(\sigma_{\mathsf{rand}})}}}$$

where $\mathbf{r} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}^m+\mathbf{c},\sigma_{\mathsf{rand}}}$, $\mathbf{A} \leftarrow_\$ \mathbb{Z}_Q^{n\times m}$ and $\mathbf{u} \leftarrow_\$ \mathbb{Z}_Q^n$.

In Supplementary Material A we recall the background for the proof of this lemma.

Proof. From Lemma 13 we have that

$$\Delta\big((\mathbf{A}\mathbf{r}, \mathbf{A}, \mathbf{e}^\top\mathbf{r}), (\mathbf{u}, \mathbf{A}, \mathbf{e}^\top\mathbf{r})\big) \leq \frac{1}{2}\sqrt{2^{n\log(Q)} \cdot 2^{-\tilde{\mathbf{H}}_\infty(\mathbf{r}|\mathbf{e}^\top\mathbf{r})}},$$

and from Lemma 14 we have that

$$\frac{1}{2}\sqrt{2^{n\log(Q)} \cdot 2^{-\tilde{\mathbf{H}}_\infty(\mathbf{r}|\mathbf{e}^\top\mathbf{r})}} \leq \frac{1}{2}\sqrt{2^{(n+1)\log(Q)} \cdot 2^{-\mathbf{H}_\infty(\mathbf{r})}}$$

because $\tilde{\mathbf{H}}_\infty(\mathbf{r}|\mathbf{e}^\top\mathbf{r}) \geq \mathbf{H}_\infty(\mathbf{r}) - \log(Q)$ since $\mathbf{e}^\top\mathbf{r}$ takes values in $\mathbb{Z}_Q$ (see Lemma 14 in Supplementary Material A).

What is left is to analyse the min entropy $\mathbf{H}_\infty(\mathbf{r})$. Note that for any $\mathbf{x} \in \mathbb{Z}^m$ we have that $\rho_{\sigma_{\mathsf{rand}}}(\mathbf{x}) \leq \rho_{\sigma_{\mathsf{rand}}}(\mathbf{0}) = 1$. Furthermore, from Lemma 16 we have that $\rho_{\sigma_{\mathsf{rand}}}(\mathbb{Z}^m + \mathbf{c}) \geq (1-\epsilon)\frac{\sigma_{\mathsf{rand}}^m}{\mathsf{det}(\mathbb{Z}^m)}$, where $\epsilon > 0$ and and assuming that $\sigma_{\mathsf{rand}}^m \geq \eta_\epsilon(\mathbb{Z}^m)$. From the fact that $\mathbf{I}_m$ is the basis of $\mathbb{Z}^m$ we have that $\mathsf{det}(\mathbb{Z}^m) = \mathsf{det}(\mathbf{I}_m) = 1$, and from Lemma 15 we have that $\eta_\epsilon(\mathbb{Z}^m) \leq C_{\epsilon,m}$.

Putting the above together we have that for all $\mathbf{x} \in \mathbb{Z}^m$

$$\begin{aligned}\mathcal{D}_{\mathbb{Z}^m+\mathbf{c},\sigma_{\mathsf{rand}}}(\mathbf{x}) &\leq \frac{\rho_{\sigma_{\mathsf{rand}}}(\mathbf{x})}{\rho_{\sigma_{\mathsf{rand}}}(\mathbb{Z}^m+\mathbf{c})}\\ &\leq \frac{1}{\rho_{\sigma_{\mathsf{rand}}}(\mathbb{Z}^m+\mathbf{c})}\\ &\leq \frac{1}{(1-\epsilon)\cdot\sigma_{\mathsf{rand}}^m}\end{aligned}$$

Then from Definition 4 we have

$$\mathbf{H}_\infty(\mathbf{r}) \geq -\log\left(\frac{1}{(1-\epsilon)\cdot\sigma_{\mathsf{rand}}^m}\right) = \log(1-\epsilon) + m\log(\sigma_{\mathsf{rand}}).$$

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* $\sigma_{rand} \geq C_{\epsilon,h}$ *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} *is a bound on the infinity norm of the noise terms in the blind rotation key* br. *Then we have*

$$\Delta(\mathbf{c}_{\mathsf{out}}, \mathbf{c}_{\mathsf{fresh}}) \leq \max\left(2\delta, \frac{1}{2}\sqrt{\frac{2^{(N+1)\log(Q)}}{2^{\log(1-\epsilon)+h\log(\sigma_{rand})}}}\right),$$

where $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$, $b_{\mathsf{fresh}} = \langle\mathbf{a}_{\mathsf{fresh}}, \mathbf{s}'\rangle + f(m) + e_{rand} + e_{\mathsf{out}}$, $e_{\mathsf{out}} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}\cdot\sqrt{1+||\mathbf{e}||}}$, $e_{rand} \leftarrow_\$ \widetilde{\mathbf{e}}^\top \cdot \mathbf{r}$, $\mathbf{r} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}^h,\sigma_{rand}}$ *and where* $\mathbf{e} \in \mathbb{Z}^{2\cdot n\cdot N\cdot\ell_{\mathsf{br}}}$ *is the vector of error coefficients in the blind rotation key, and* $\widetilde{\mathbf{e}} \in \mathbb{Z}^h$ *are the error terms in the vector of LWE samples* $\mathbf{v}$.

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

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

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 Steps 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

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

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

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

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 $\mathbf{c}_\mathsf{out} \leftarrow \mathsf{extProd}_\mathsf{simul}(\mathbf{C}_G, \mathbf{d} - \mathbf{h}) + \mathbf{h}$, 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 bit $\mathfrak{m}_G \in \{0, 1\}$. 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 $\mathbf{d} - \mathbf{h}$ 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 $\mathfrak{e}_d = \mathfrak{e}_h \cdot X^l \in \mathcal{R}_Q$.

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 $b_\mathsf{ext} = \mathfrak{b}_\mathsf{acc}[1] = \langle \mathbf{a}_\mathsf{acc}, \mathbf{s}' \rangle + \mathfrak{a}_\mathsf{rot} \cdot X^{\mathsf{Phase}(\mathbf{c})}[1] + (\sum_{i=1}^{n} \widehat{\mathfrak{e}}_i \cdot X^{\sum_{j=i+1}^{n} -\mathbf{a}[j] \cdot \mathbf{s}[j]})[1]$. 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 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 $\mathbf{c}_{\mathsf{out}} = \mathbf{c}_{\mathsf{ext}} + \mathbf{r}^{\top} \cdot \mathbf{v} + y$ 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 $\mathbf{c}_{\mathsf{rand}} \leftarrow \mathbf{v}^{\top} \cdot \mathbf{r}$ with $\mathbf{r} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}^h, \sigma_{\mathsf{rand}}}$, $y \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}, \sigma_{\mathsf{x}}}$ and $\mathbf{v}$ is a vector of size h 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 $b_{\mathsf{rand}} = \langle \mathbf{a}_{\mathsf{rand}}, \mathbf{s}' \rangle + e_{\mathsf{rand}}$ where $e_{\mathsf{rand}} \leftarrow_{\$} \widetilde{\mathbf{e}}^{\top} \cdot \mathbf{r}$ with $\mathbf{r} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}^h, \sigma_{\mathsf{rand}}}$ and where $\widetilde{\mathbf{e}}$ denotes the errors of the LWE samples in $\mathbf{v}$.

Claim. Given that $\sigma_{\mathsf{rand}} \geq C_{\epsilon,h}$ the statistical distance between Hybrid 2 and Hybrid 1 is at most

$$\frac{1}{2}\sqrt{\frac{2^{(N+1)\log(Q)}}{2^{\log(1-\epsilon)+h\log(\sigma_{\mathsf{rand}})}}}$$

for some $\epsilon > 0$.

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

Now it is easy to see that the rest of the proof follows directly by applying Lemma 10. Note that the notation in Lemma 10 is mostly already in place, except that we set the m from the lemma to h and the n from the lemma to N. 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. Note that Lemma 10 requires Q to be an odd prime. In Remark 2 we discuss how we handle a non-prime Q. Finally, the uniform vector $\mathbf{u}$ from the lemma corresponds to $\mathbf{a}_{\mathsf{rand}}$.

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 $e_{\mathsf{out}} \leftarrow_{\$} \mathcal{D}_{\mathbb{Z}, \sigma_{\mathbf{x}} \cdot \sqrt{1+||\mathbf{e}||}}$. In particular, the noise term is independent from the 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 8 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

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

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 8 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 1 (On FHEW Blind Rotation.). Theorem 1 works for the blind rotation algorithm from [CGGI16a]. 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 analogous theorem with the blind rotation from FHEW [DM15] because it is a sequence of external products. Consequently, we can omit the "Single MUX Gate" step in the proof of Theorem 1. We recall FHEW [DM15] and give the proof in Supplementary Material D.

Proof of Circuit Privacy. Remind that according to Definition 2, to prove circuit privacy we have to show a simulator that, on input ek and $m_{\mathsf{out}} = \mathcal{C}(m_1, \ldots, m_n)$ 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, bootstrapping in simul mode and adding m_{out} to the resulting ciphertext. 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. Denote as $\mathbf{c}_{\mathsf{out}}$ the ciphertext returned by Eval. Recall that $\mathbf{c}_{\mathsf{out}}$ is distributed as given by Lemma 1, because Eval ends with an invocation of $\mathsf{Bootstrap}$ in simulation mode. Set $\mathbf{c} = \mathbf{0}$ and $\mathfrak{a}_{\mathsf{rot}} = \mathfrak{o}$. The simulator runs and outputs

$$\mathbf{c}_{\mathsf{fresh}} \leftarrow \mathsf{Bootstrap}_{\mathsf{simul}}(\mathsf{br}, \mathsf{ksK}, \mathbf{c}, \mathfrak{a}_{\mathsf{rot}}, \mathbf{v}, \sigma_{\mathsf{rand}}, \sigma_{\mathbf{x}}) + [\mathbf{0}, m_{\mathsf{out}}].$$

Denote $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$. From Lemma 1 we have that $\mathbf{c}_{\mathsf{out}}$ is distributed as in Eval. Namely, $\mathbf{a}_{\mathsf{fresh}}$ is statistically close to uniform, $b_{\mathsf{fresh}} = \langle \mathbf{a}_{\mathsf{fresh}}, \mathbf{s}' \rangle + m_{\mathsf{out}} + e_{\mathsf{rand}} + e_{\mathsf{out}}$ with $e_{\mathsf{rand}} + e_{\mathsf{out}}$ distributed as in Lemma 1. Hence the $\mathbf{c}_{\mathsf{fresh}}$ is statistically close to $\mathbf{c}_{\mathsf{out}}$, and in particular independent from the circuit $\mathcal{C}$.

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 [DS16]. Remind that Ducas and Stehlé [DS16] 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.

Remark 2. In the case of a power of two modulus Q, we slightly modify the scheme and choose the LWE samples with respect to a prime modulus that is slightly larger than Q. Then, after Step 8 of Algorithm 3 we switch the modulus to Q. This change is made because the leftover hash lemma (Lemma 10) requires a universal hash function, which is satisfied if the modulus is prime but not when the modulus is a power of two.

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. Remind that, in contrast to computational security, the advantage in breaking a property for statistical security does not increase with advances in high-performance computing. Furthermore, since we measure the distance between samples, the ability to distinguish two distributions depends on the number of samples given.

The parameters are listed in Table 1. We estimate the (R)LWE security using the latest commit of the LWE estimator [APS15]. 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 [DS16] that we refer to as DS-WM. For completeness we recall the relevant lemmas from [DS16] that we used for our estimations in Supplementary Material A. 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 [APS15], (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 (see Supplementary Material C for estimations). 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 $\mathsf{L}_{\mathsf{Bk}}\text{-}\mathsf{simul}$, (3) $\mathsf{L}_{\mathsf{Bk}}\text{-}\mathsf{det} = \mathsf{L}_{\mathsf{Bk}}\text{-}\mathsf{simul}^k$ 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.

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

Method \ $\log_2$(Param.)	Q	L_{Bk}, ℓ_{Bk}-simul	L_{Bk}, ℓ_{Bk}-det	σ_{Ksk}	σ_{x}	U_x	σ_{rand}	h	Δ
Ours-Int	48	8, 6	24, 2	26	17.7	12.8	-	12.3	80
DS-WM-Int	48	8, 6	24, 2	26	-	41.2	12.3	12.8	16
Ours-Double	36	4, 9	12, 3	14	8.9	-	21.9	11.7	80
DS-WM-Double	36	4, 9	12, 3	14	-	30.8	21.9	11.7	6.6

Table 1. Parameter Sets. We list base-two logarithms of the parameters. For all sets we set $n = 912$, $N = 2^{11}$, $\mathsf{L}_{\mathsf{Ksk}} = 2^7$ and $\sigma_{\mathsf{br}} = 3.2$. Note that the column L_{Bk}-simul gives the decomposition base for $\mathsf{ver} = \mathsf{simul}$ and L_{Bk}-det for $\mathsf{ver} = \mathsf{det}$. 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 5 and 12 times, respectively.

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 [DM15, CGGI16a, CGGI20] 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. 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 [DS16] propose to instantiate their washing machine method on a parameter set from the Ducas, and Micciancio's FHEW bootstrapping [DM15], 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 32 bit modulus and set all decomposition bases to 2 (smallest and least efficient possible) to maximize correctness. We set $n = 810$, and the LWE standard deviations to 3.2, which gives us a low-security level of only 84 bits according to the LWE estimator [APS15]. Our estimates show that the correctness of $\mathbf{c}_{\mathsf{in}}$ was around the 2^{-33} level already for deterministic computation. We noticed that increasing the modulus Q, and hence dropping LWE security below 80 bits, does not change the correctness 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 [DM15]. To conclude, our analysis shows that the parameter choice in [DS16] for DS-WM cannot give circuit privacy is 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 [CGGI16a, CGGI20] in simulation mode due to the small ring degree with statistical security larger than 30-bits.

	Ours-Int			
t	det		simul	
	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$	$\mathbf{c}_{\mathsf{out}}$	$\mathbf{c}_{\mathsf{in}}$
4	-771	-239	-494	-154
5	-495	-99	-317	-64
6	-345	-40	-221	-26
		$\vdots$		
10	-126	0.90	-82	0.82
11	-105	0.99	-68	0.97

	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	-771	-239	-198	-198
5	-495	-99	-69	-69
6	-345	-40	-20	-20
		$\vdots$		
10	-126	0	0	0
11	-105	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	-841	-260	-307	-96
5	-540	-108	-198	-41
6	-376	-43	-138	-17
		$\vdots$		
10	-137	0	-51	0
11	-114	0	-41	0

	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	-841	-260	-96	-96
5	-613	-108	-38	-38
6	-376	-43	-14	-14
		$\vdots$		
10	-137	0	0	0
11	-114	0	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.

	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		1.36		1.36	79%						
Ours-Int-C	0.14	1.01	0.14	1.01	72%	79	104	134	179	186	248
DS-WM-Int		2.03		0.39	–						
Ours-Double		1.33		1.33	59%						
Ours-Double-C	0.27	0.91	0.27	0.91	40%	56	89	168	268	69	110
DS-WM–Double		7.10		0.59	–						

Table 3. Performance. The BL and KS columns give the blind rotation and key switching timings. The suffix "-C" in the parameter sets stands for using rounded continuous Gaussian sampling. 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. To implement negacyclic convolutions for Ours-Int and DS-WM-Int, we used the Intel Hexl library [BKS+21]. 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 [FJ21] 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 [MP12]. For general moduli, which is the case for Our-Int and DM-WM-Int, we implement a version of the method by Genise and Micciancio [GM18, CDCG+18]. Both samplers are instantiated with either the Karney method [Kar16] to sample for the exact discrete Gaussian distribution, or (somewhat heuristically) we use the Box-Muller transform [BM58] with rounding to the closest integer. The implementation using the Box-Muller transform serves mostly for comparison and to showcase potential speedups. Our proofs require a discrete Gaussian sampler for simplicity. Nevertheless, we note that in some cases, proofs can be generalized [HLS17] to support rounded continuous Gaussian distributions while preserving security. We leave such generalizations as future work.

As we can see from Table 3, our method is faster when compared to both DS-WM methods. In particular, Ours-Int is roughly 1.56× faster than DS-WM-Int

(2.03× for rounded Box-Muller). Then, Our-Double is about 5.46× faster than its DS-WM-Double analog (7.1× for rounded Box-Muller).

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 could 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].

Finally, note that in the case of Our-Int, sanitization is roughly 9× slower (7× for rounded Box-Muller) than deterministic computation. For Our-Double, sanitization is only 4.8× slower (3.3× for rounded Box-Muller) 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 a straightforward implementation of the method from [GM18, CDCG$^+$18]. 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. Furthermore, 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 outperforms the Ducas-Stehlé washing machine method [DS16].

Importantly, our experiments show that 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. Furthermore, using faster discrete Gaussian sampling algorithms [MW17, DFW22] instead of Karney [Kar16] may further improve the performance. Finally, an interesting problem is to analyze whether we can use randomized gadget decomposition that has output from other distributions and discrete Gaussian. In particular, it may be worth exploring the use of more efficient subgaussian samplers [GMP19, ZY22, JLP21] in place of the Gaussian sampling algorithms [GM18, CDCG$^+$18].

References

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

ACC$^+$18. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

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

AGHS13. Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete Gaussian leftover hash lemma over infinite domains. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 97–116. Springer, Heidelberg, December 2013.

AGHV22. Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 70–99. Springer, Heidelberg, November 2022.

AJL$^+$12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, Heidelberg, April 2012.

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

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, Heidelberg, August 2014.

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

AR13. Divesh Aggarwal and Oded Regev. A note on discrete gaussian combinations of lattice vectors, 2013.

BDPMW16. Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. Fhe circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 62–89, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

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

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.

BI22. Florian Bourse and Malika Izabachène. Plug-and-play sanitization for TFHE. Cryptology ePrint Archive, Report 2022/1438, 2022. https://eprint.iacr.org/2022/1438.

BIP$^+$18. Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 699–729. Springer, Heidelberg, November 2018.

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

BM58. George EP Box and Mervin E Muller. A note on the generation of random normal deviates. *The annals of mathematical statistics*, 29(2):610–611, 1958.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106. IEEE Computer Society Press, October 2011.

BV14. Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*, pages 1–12. Association for Computing Machinery, January 2014.

CDCG+18. David Bruce Cousins, Giovanni Di Crescenzo, Kamil Doruk Gür, Kevin King, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, and Erkay Savas. Implementing conjunction obfuscation under entropic ring lwe. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 354–371, 2018.

CDKS19. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 395–412. ACM Press, November 2019.

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

CGGI16a. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Heidelberg, December 2016.

CGGI16b. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

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

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 315–337. Springer, Heidelberg, April / May 2018.

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

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 670–699. Springer, Heidelberg, December 2021.

CLR17. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1243–1255. ACM Press, October / November 2017.

CO17. Wutichai Chongchitmate and Rafail Ostrovsky. Circuit-private multi-key FHE. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 241–270. Springer, Heidelberg, March 2017.

DFW22. Yusong Du, Baoying Fan, and Baodian Wei. An improved exact sampling algorithm for the standard normal distribution. *Computational Statistics*, 37(2):721–737, 2022.

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

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

DORS08. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. 38(1):97–139, mar 2008.

DRS04. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, Heidelberg, May 2004.

DS16. Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 294–310. Springer, Heidelberg, May 2016.

DSGKS21. Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. Towards a ring analogue of the leftover hash lemma. *Journal of Mathematical Cryptology*, 15(1):87–110, 2021.

FJ21. Matteo Frigo and Steven G. Johnson. Fftw. https://www.fftw.org, 2021.

Gen09a. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.

GH11. Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 107–109. IEEE Computer Society Press, October 2011.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, Heidelberg, May 2012.

GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172. Springer, Heidelberg, August 2010.

GM18. Nicholas Genise and Daniele Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 174–203. Springer, Heidelberg, April / May 2018.

GMP19. Nicholas Genise, Daniele Micciancio, and Yuriy Polyakov. Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 655–684. Springer, Heidelberg, May 2019.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.

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

HFH99. Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 78–86, New York, NY, USA, 1999. Association for Computing Machinery.

HLS17. Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded Gaussians – fast and secure constant-time sampling for lattice-based crypto. Cryptology ePrint Archive, Report 2017/1025, 2017. https://eprint.iacr.org/2017/1025.

HS15. Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, Heidelberg, April 2015.

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

IP07. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, Heidelberg, February 2007.

JKLS18. Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1209–1222. ACM Press, October 2018.

JLP21. Sohyun Jeon, Hyang-Sook Lee, and Jeongeun Park. Efficient lattice gadget decomposition algorithm with bounded uniform distribution. *IEEE Access*, 9:17429–17437, 2021.

JVC18. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 1651–1669. USENIX Association, August 2018.

Kar16. Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1), jan 2016.

Klu22. Kamil Kluczniak. NTRU-v-um: Secure fully homomorphic encryption from NTRU with small modulus. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on*

Computer and Communications Security, pages 1783–1797. ACM Press, November 2022.

KS22. Kamil Kluczniak and Leonard Schild. Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, Nov. 2022.

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

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

Lat22. Lattigo v4. Online: https://github.com/tuneinsight/lattigo, August 2022. EPFL-LDS, Tune Insight SA.

LJLA17. Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 619–631. ACM Press, October / November 2017.

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

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

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer, Heidelberg, May 2013.

Mea86. Catherine Meadows. 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*, pages 134–134, 1986.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, Heidelberg, April 2012.

MP21. Daniele Micciancio and Yuriy Polyakov. *Bootstrapping in FHEW-like Cryptosystems*, page 17–28. Association for Computing Machinery, New York, NY, USA, 2021.

MR04. D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 372–381, 2004.

MW17. Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, Heidelberg, August 2017.

Nui22. Koji Nuida. How to handle invalid queries for malicious-private protocols based on homomorphic encryption. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, APKC '22, page 15–25, New York, NY, USA, 2022. Association for Computing Machinery.

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

RAD78. RL Rivest, L Adleman, and ML Dertouzos. On data banks and privacy homomorphisms. foundations of secure computation (1978), 169–180. *Search in*, 1978.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005.

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

RSC$^+$19. M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1501–1518. USENIX Association, August 2019.

SS11. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, Heidelberg, May 2011.

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

ZY22. Shiduo Zhang and Yang Yu. Towards a simpler lattice gadget toolkit. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 498–520. Springer, Heidelberg, March 2022.

Supplementary Material

A Additional Preliminaries

In this section we recall some useful lemmas and definitions.

Definition 3 (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*

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

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

A.1 Probability Theory

Lemma 11 (Smudging Lemma [AJL+12]). *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 12 (Lemma 2.3 from [DS16]). *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.

Definition 4. *The min-entropy of a random variable X is defined as*

$$\mathbf{H}_\infty(X) = -\log\left(\max_x \Pr[X = x]\right)$$

Furthermore we recall the definition of average min-entropy A given B as

$$\tilde{\mathbf{H}}_\infty(A|B) = -\log\left(\mathbb{E}_{b \leftarrow B}\big[\max_a \Pr[A = a|B = b]\big]\right) = -\log\left(\mathbb{E}_{b \leftarrow B}[2^{-\mathbf{H}_\infty(A|B=b)}]\right).$$

Lemma 13 (Generalized Leftover Hash Lemma [DRS04, DORS08]). *Assume $\{\mathsf{H}_X \; \{0,1\}^n \mapsto \{0,1\}^\ell\}_{x \in \mathcal{X}}$ is a family of universal hash functions. Then, for any random variables W and I,*

$$\Delta\big((\mathsf{H}_X(W), X, I), (U_\ell, X, I)\big) \leq \frac{1}{2} \cdot \sqrt{2^\ell \cdot 2^{-\tilde{\mathbf{H}}_\infty(W|I)}}$$

Lemma 14 (Lemma 2.2. in [DRS04, DORS08]). *Let A, B and C be random variables. Then*

1. *For any $\delta > 0$, the conditional entropy $\mathbf{H}_\infty(A|B=b)$ is at least $\tilde{\mathbf{H}}_\infty(A|B) - \log(1/\delta)$ with probability at least $1-\delta$ over the choice of b.*
2. *If B has at most 2^λ possible values, then*
$$\tilde{\mathbf{H}}_\infty(A|(B,C)) \geq \tilde{\mathbf{H}}_\infty((A,B)|C) - \lambda \geq \tilde{\mathbf{H}}_\infty(A|C) - \lambda.$$
In particular, $\tilde{\mathbf{H}}_\infty(A|B) \geq \mathbf{H}_\infty\big((A,B)\big) - \lambda \geq \mathbf{H}_\infty(A) - \lambda$.

A.2 Lattices

Definition 5 (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 15 ([MR04], 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 $C_{\delta,m} = \sqrt{\frac{\ln(2m(1+1/\delta))}{\pi}}$.

Lemma 16 (Claim 3.8 in [Reg09]). *For any lattice Λ, $\mathbf{c} \in \mathbb{R}^n$, $\epsilon > 0$ and $\sigma \geq \eta_\epsilon$,*
$$\rho(\Lambda + \mathbf{c}) \in \frac{\sigma^n}{\mathsf{det}(\Lambda)}(1 \pm \epsilon)$$

Lemma 17 (Corollary 2.8 in [GPV08]). *Let $\Lambda \subseteq \mathbb{Z}^m$ be a lattice, $0 < \epsilon < 1$, $\sigma > 0$. For any vector $\mathbf{c} \in \mathbb{R}^m$, if $\sigma \geq \eta_\epsilon(\Lambda)$, then we have*
$$\rho(\Lambda + \mathbf{c}) \in \left[\frac{1-\epsilon}{1+\epsilon}, 1\right] \cdot \rho_\sigma(\Lambda)$$

A.3 Gaussian Sampling Algorithms

We recall the Gaussian sampling algorithm form [GM18] on Figure 5. For completeness, we also recall the Gaussian sampling algorithm from [MP12] on Figure 4 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.

```
$\mathsf{G}^{-1}_{\mathsf{simul}}(a, \mathsf{L}, \sigma_{\mathbf{x}})$:
Input:
Integers $a \in \mathbb{Z}_Q$ and $\mathsf{L}$ 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 :     $\mathbf{x}[j] = \mathcal{D}_{\mathsf{L}\mathbb{Z}+a,\sigma_{\mathbf{x}}}$.
3 :     $a \leftarrow (a - \mathbf{x}[i][j])/\mathsf{L}$.
4 : Output $\mathbf{x}$.
```

Fig. 4. Gaussian Sampling Algorithm [MP12].

In practice, however, a modulus of the form $Q = \mathsf{L}^\ell$ forces us to implement 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 5 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 [GM18]. furthermore, we refer to [MP12] for the analysis of the Gaussian sampling algorithm on Figure 4 for the special case $Q = \mathsf{L}^\ell$.

B Missing Proofs

Now we give our generalization of Lemma 3.6 from [BDPMW16], which itself is an adaptation of lemma 3.3 from [AR13]. The proof is essentially a copy-paste from the proof in [BDPMW16], except that we use a different bound on the smoothing parameter that we showed in Lemma 9.

Lemma 8 (Gaussian Leftover Hash Lemma (Generalized Lemma 3.6 from [BDPMW16])). *Let δ, $\sigma_{\mathbf{x}} > 0$. Let $\mathcal{L} = \{\mathbf{v} = \hat{\Lambda} : \hat{\mathbf{e}}^\top \cdot \mathbf{v} = 0\}$, where $\hat{\mathbf{e}} = [\mathbf{e}, 1] \in \mathbb{Z}^{m+1}$, $\hat{\Lambda} = \hat{\Lambda}^\perp_Q(\mathbf{G}_{\mathsf{L},Q,w}) \times \mathbb{Z}$, $m = w \cdot \ell$ 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 $\mathbf{x} \leftarrow_\$ \mathcal{D}_{\Lambda^\perp_Q(\mathbf{G}_{\mathsf{L},Q,w})+\mathbf{c},\sigma_{\mathbf{x}}}$, $y \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}}}$, $e' \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}} \cdot \sqrt{1+||\mathbf{e}||^2}}$, 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 $e' \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\sigma_{\mathbf{x}} \cdot \sqrt{1+m\sigma^2_{\mathsf{br}}}}$.

```
$\mathsf{G}^{-1}_{\mathsf{simul}}(a, \mathsf{L}, \sigma_{\mathbf{x}})$:
Input:
Integers $a \in R_Q$ and $\mathsf{L}$ s.t. $Q < \mathsf{L}^\ell$, and a Gaussian parameter $\sigma_{\mathbf{x}}$.
1 : Set $\mathbf{q} \leftarrow \mathsf{G}^{-1}_{\mathsf{det}}(Q, \mathsf{L})$ and $\mathbf{u} \leftarrow \mathsf{G}^{-1}_{\mathsf{det}}(a, \mathsf{L})$.
2 : $\sigma = \sigma_{\mathbf{x}}/(\mathsf{L}+1)$.
3 : $\mathbf{p} \leftarrow \mathsf{Perturb}(\sigma, \sigma_{\mathbf{x}})$.
4 : For each $i \in [\ell]$
5 :     $\mathbf{c}[i] \leftarrow (\mathbf{c}[i-1] - \mathbf{u}[i] - \mathbf{p}[i])/\mathsf{L}$.
6 : $\mathbf{z} \leftarrow \mathsf{SampleD}(\sigma, \mathbf{c}, \sigma_{\mathbf{x}})$.
7 : For each $i \in [\ell - 1]$
8 :     $\mathbf{t}[i] \leftarrow \mathsf{L} \cdot \mathbf{z}[i] - \mathbf{z}[i-1] + \mathbf{q}[i] \cdot \mathbf{z}[\ell - i] + \mathbf{u}[i]$.
9 : $\mathbf{t}[\ell] \leftarrow \mathbf{q}[\ell] \cdot \mathbf{z}[\ell] - \mathbf{z}[\ell-1] + \mathbf{u}[\ell]$.
10 : Return $\mathbf{t}$.

$\mathsf{SampleD}(\sigma, \mathbf{c}, \sigma_{\mathbf{x}})$:
Input:
Gaussian parameters $\sigma$ and $\sigma_{\mathbf{x}}$, and a vector $\mathbf{c} \in \mathbb{R}^\ell$.
1 : $\mathbf{z}[\ell] \leftarrow \lfloor -\mathbf{c}[\ell]/\mathbf{d}[\ell] \rfloor$.
2 : $\mathbf{z}[\ell] \leftarrow \mathbf{z}[\ell] + \mathsf{SampleZ}_t(\sigma/\mathbf{d}[\ell], \lfloor -\mathbf{c}[\ell]/\mathbf{d}[\ell] \rfloor_{[0,1)}, \sigma_{\mathbf{x}})$.
3 : $\mathbf{c} \leftarrow \mathbf{c} - \mathbf{z}[\ell] \cdot \mathbf{d}$.
4 : For all $i \in [\ell - 1]$.
5 :     $\mathbf{z}[i] \leftarrow \lfloor -\mathbf{c}[i] \rfloor + \mathsf{SampleZ}_t(\sigma, \lfloor -\mathbf{c}[i] \rceil_{[0,1)}, \sigma_{\mathbf{x}})$.
6 : Return $\mathbf{z}$.

$\mathsf{Perturb}(\sigma, \sigma_{\mathbf{x}})$:
Input:
A Gaussian parameters $\sigma$ and $\sigma_{\mathbf{x}}$.
1 : $\beta \leftarrow 0$.
2 : For $i$ in $[\ell]$:
3 :     $\mathbf{c} \leftarrow \beta/\mathbf{l}[i]$ and $\sigma[i] \leftarrow \sigma/\mathbf{l}[i]$
4 :     $\mathbf{z} \leftarrow \lfloor \mathbf{c}[i] \rfloor + \mathsf{SampleZ}_t(\sigma[i], \lfloor \mathbf{c}[i] \rceil_{[0,1)}, \sigma_{\mathbf{x}})$.
5 :     $\beta \leftarrow -\mathbf{z}[i]\mathbf{h}[i]$.
6 : $\mathbf{p}[1] \leftarrow (2\mathsf{L}+1)\mathbf{z}[1] + \mathsf{L}\mathbf{z}[1]$.
7 : For $i$ in $[2, \ell]$:
8 :     $\mathbf{p} \leftarrow \mathsf{L}(\mathbf{z}[i-1] + 2\mathbf{z}[i] + [i+1])$.
9 : Return $\mathbf{p}$.
```

Fig. 5. Gaussian Sampling Algorithm [GM18] 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

Proof. Let $\hat{\mathbf{c}} = [\mathbf{c}, 0] \in \mathbb{Z}^{m+1}$ and $\hat{\Lambda} = \Lambda_Q^\perp(\mathbf{G}_{\mathsf{L},Q,w}) \times \mathbb{Z}$. We want to show that

$$\Delta\big(\hat{\mathbf{e}}^\top \mathcal{D}_{\hat{\Lambda}+\hat{\mathbf{c}},\sigma_{\mathbf{x}}}, \mathcal{D}_{\mathbb{Z},||\hat{\mathbf{e}}||\sigma_{\mathbf{x}}}\big) \leq 2\delta.$$

The support of $\hat{\mathbf{e}}^\top \mathcal{D}_{\hat{\Lambda}+\hat{\mathbf{c}}}$ is $\hat{\mathbf{e}}^\top \hat{\Lambda} + \hat{\mathbf{e}}^\top \hat{\mathbf{c}} = \mathbf{e}^\top \Lambda_Q^\perp(\mathbf{G}_{\mathsf{L},Q,w}) + \mathbb{Z} + \mathbf{e}^\top \mathbf{c} = \mathbb{Z}$. Fix some $z \in \mathbb{Z}$. The probability mass assigned to z by $\hat{\mathbf{e}}^\top \mathcal{D}_{\hat{\Lambda}+\hat{\mathbf{c}},\sigma_{\mathbf{x}}}$ is proportional to $\rho_{\sigma_{\mathbf{x}}}(\mathcal{L}_z)$, where

$$\mathcal{L}_z = \big\{\mathbf{v} \in \hat{\Lambda} + \hat{\mathbf{c}} : \hat{\mathbf{e}}^\top \mathbf{v} = z\big\}.$$

We define the lattice $\mathcal{L} = \{\mathbf{v} \in \hat{\Lambda} : \hat{\mathbf{e}}^\top \mathbf{v} = 0\}$; note that $\mathcal{L}_z = \mathcal{L} + \mathbf{w}_z$ for any $\mathbf{w}_z \in \mathcal{L}_z$. Let $\mathbf{u}_z = \frac{z}{||\hat{\mathbf{e}}||^2 \sigma_{\mathbf{x}}} \hat{\mathbf{e}}$. Then $\mathbf{u}_z$ is clearly proportional to $\hat{\mathbf{e}}$. Observe that $\mathbf{u}_z$ is orthogonal to $\sigma_{\mathbf{x}}^{-1} \mathcal{L}_z - \mathbf{u}_z$. Indeed for any $\mathbf{t} \in \sigma_{\mathbf{x}}^{-1} \mathcal{L}_z$ we have $\hat{\mathbf{e}}^\top(\mathbf{t} - \mathbf{u}_z) = 0$. From this we have $\rho(\mathbf{t}) = \rho(\mathbf{u}_z) \cdot \rho(\mathbf{t} - \mathbf{u}_z)$, and by summing for $\mathbf{t} \in \sigma_{\mathbf{x}}^{-1} \mathcal{L}_z$:

$$\rho(\sigma_{\mathbf{x}}^{-1} \mathcal{L}_z) = \rho(\mathbf{u}_z) \cdot \rho(\sigma_{\mathbf{x}}^{-1} \mathcal{L}_z - \mathbf{u}_z).$$

Observe that we have $\sigma_{\mathbf{x}}^{-1} \mathcal{L}_z - \mathbf{u}_z = \sigma_{\mathbf{x}}^{-1}(\mathcal{L} - \mathbf{c}')$ for some $\mathbf{c}'$ in the vector span of the lattice $\mathcal{L}$ (because $\mathcal{L}_z - \sigma_{\mathbf{x}}^{-1} \mathbf{u}_z = \mathcal{L} + \mathbf{w}_z - \sigma_{\mathbf{x}} \mathbf{u}_z$ and $\hat{\mathbf{e}}^\top(\mathbf{w}_z - \sigma_{\mathbf{x}} \mathbf{u}_z) = 0$). Then using[4] Lemma 17 and Lemma 9 that bounds $\sigma_{\mathbf{x}}$ as in the theorem statement, we obtain

$$\begin{aligned}
\rho(\sigma_{\mathbf{x}}^{-1} \mathcal{L}_z) &= \rho(\mathbf{u}_z) \cdot \rho_{\sigma_{\mathbf{x}}}(\mathcal{L} - \mathbf{c}') \\
&\in \left[\frac{1-\delta}{1+\delta}, 1\right] \cdot \rho_{\sigma_{\mathbf{x}}}(\mathcal{L}) \cdot \rho(\mathbf{u}_z) \\
&\in \left[\frac{1-\delta}{1+\delta}, 1\right] \cdot \rho_{\sigma_{\mathbf{x}}}(\mathcal{L}) \cdot \rho\Big(\frac{z}{||\hat{\mathbf{e}}||^2 \sigma_{\mathbf{x}}} \hat{\mathbf{e}}\Big) \\
&\in \left[\frac{1-\delta}{1+\delta}, 1\right] \cdot \rho_{\sigma_{\mathbf{x}}}(\mathcal{L}) \cdot \rho_{||\hat{\mathbf{e}}||\sigma_{\mathbf{x}}}.
\end{aligned}$$

This implies that the statistical distance between $\hat{\mathbf{e}}^\top \mathcal{D}_{\hat{\Lambda}+\hat{\mathbf{c}},\sigma_{\mathbf{x}}}$ and $\mathcal{D}_{\mathbb{Z},||\hat{\mathbf{c}}||\sigma_{\mathbf{x}}}$ is at most $1 - \frac{1-\delta}{1+\delta} \leq 2\delta$.

C Error Analysis and Missing Algorithms

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

[4] This is the place where our proof differs from [BDPMW16]. Namely, we use our Lemma 9 to give a different bound on $\sigma_{\mathbf{x}}$.

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

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

The following $\sigma_1^2 \leq 2\ell_{\mathsf{br}} \cdot N \cdot \sigma_G^2 \cdot \mathsf{B}(\mathsf{G}_{\mathsf{ver}}^{-1}(., \mathsf{L}_{\mathsf{br}}))$ 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, $\sigma_{\mathsf{out}}^2 \leq \sigma^2 + \mathfrak{m}_G \cdot \sigma_1^2$ 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 $\sigma_{\mathsf{out}}^2 \leq 2\ell_{\mathsf{br}} \cdot N \cdot \sigma_G^2 \cdot \mathsf{Var}(\mathsf{G}_{\mathsf{ver}}^{-1}(., \mathsf{L}_{\mathsf{br}})) + \sigma^2$, 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:

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

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

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

The expectation of the output noise satisfies

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

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

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

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

```
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. 6. 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 $\mathbf{c} = [\mathbf{a}, b]$, 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 $\mathsf{br} = \mathsf{RGSW}_{\sigma_{\mathsf{br}}}(\mathfrak{s},\ .)^n$.
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 : $\mathbf{c}_{\mathsf{msb},\mathsf{ksK}} \leftarrow \mathsf{KeySwitch}(\mathbf{c}_{\mathsf{msb}}, \mathsf{ksK}) \in \mathbb{Z}_Q^{n+1}$.
6 : $\mathbf{c}_{\widehat{\mathsf{msb}}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{\mathsf{msb},\mathsf{ksK}}, 2N) \in \mathbb{Z}_{2N}^{n+1}$.
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. 7. Bootstrapping: The full domain functional bootstrapping from [YXS+21, LMP21]. 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 $h \cdot \sigma_R^2 \cdot \sigma_{\mathsf{rand}}^2$ 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 $\mathbf{c}_{\mathsf{pre}} = [\mathbf{a}_{\mathsf{pre}}, b_{\mathsf{pre}}]$ 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 choice). We measured the numerical error when computing products of ring elements with the fftw library [FJ21]. The result is depicted in Table 4. Based on this table, we ruled out choosing certain moduli and decomposition bases while preserving correctness.

Note that Table 4 gives only the error of polynomial multiplication. Note that the impact on the ciphertext error of an external product is much higher and dependent on the secret norm of the secret key. Let $(\mathfrak{a}, \mathfrak{b})$ by a RLWE ciphertext such that $\mathfrak{b} - \mathfrak{a} \cdot \mathfrak{s} = \mathfrak{e}$, where $\mathfrak{e}$ is small. Denote

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

$\log_2(B)$ \ $\log_2(Q)$	32	34	36	38	40	42	44	46	48
4	0	0	0	0.5	1.14	5.5.	17.3	77.7	346.91
6	0	0	0.23	1.25	4.1	24.1	72.4	325.9	-
8	0	0.5	1.9	4.2	17.5	108.1	263.3	-	-
10	0.6	1.4	4.1	19.1	74.7	290.2	-	-	-
12	1.3	3.9	19.0	78.1	326	-	-	-	-
14	4.6	17.8	68.2	505.7	-	-	-	-	-
16	15.8	87.8	282.2	-	-	-	-	-	-
18	78.2	490.9	-	-	-	-	-	-	-

Table 4. The standard deviation of the numerical errors when multiplying degree $N = 2^{11}$ polynomials. One polynomial has coefficient modulus Q, and the other one has coefficient modulus B. We choose both polynomials uniformly at random. We denote by "-" when the $\log_2(Q) + \log_(B) + \log +2(N) \geq 64$, in which case we exceed 64 bits and end up with random-looking polynomials.

where $\mathfrak{r}_1$ and $\mathfrak{r}_2$ is the numerical error introduce by the multiplication algorithm **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

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

D Circuit Privacy for FHEW-style Blind Rotation

We recall the FHEW blind rotation algorithm [DM15] at Figure 8. 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 $\boxed{\text{red box}}$.

```
$\mathsf{BRKeyGen}(\sigma_{\mathsf{br}}, \mathsf{L}_{\mathsf{FHEW}}, \mathfrak{s}, \mathbf{s})$:
**Input:**
An error distribution $\sigma_{\mathsf{br}}$.
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 $\mathsf{simul}$] 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. 8. FHEW-style Blind Rotation and its Setup.

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* $\mathsf{Bootstrap}$ *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* $\mathsf{Bootstrap}$ *algorithm. Let* $\mathbf{c}_{\mathsf{out}}$ *be the LWE sample returned by the* $\mathsf{Bootstrap}$ *algorithm for* $\mathsf{ver} = \mathsf{simul}$ *and Gaussian parameters* σ_{rand} *and* $\sigma_{\mathbf{x}}$ *where the Gaussian sampling algorithm* $\mathsf{G}^{-1}_{\mathsf{simul}}$ *is as in Lemma 1. Assume that* $\sigma_{rand} \geq C_{\epsilon,h}$ *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*

$$\Delta(\mathbf{c}_{\mathsf{out}}, \mathbf{c}_{\mathsf{fresh}}) \leq \max\left(2\delta, \frac{1}{2}\sqrt{\frac{2^{(N+1)\log(Q)}}{2^{\log(1-\epsilon)+h\log(\sigma_{rand})}}}\right),$$

where $\mathbf{c}_{\mathsf{fresh}} = [\mathbf{a}_{\mathsf{fresh}}, b_{\mathsf{fresh}}]$, $b_{\mathsf{fresh}} = \langle \mathbf{a}_{\mathsf{fresh}}, \mathbf{s}' \rangle + f(m) + e_{rand} + e_{\mathsf{out}}$, $e_{rand} \leftarrow_\$ \widetilde{\mathbf{e}}^\top \cdot \mathbf{r}$, $\mathbf{r} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}^h, \sigma_{rand}}$ *and* $\widetilde{\mathbf{e}} \in \mathbb{Z}^h$ *are the error terms in the vector of LWE samples* $\mathbf{v}$*. And finally* $\boxed{e_{\mathsf{out}} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}, \sigma_{\mathbf{x}} \sqrt{1 + 2n\ell_{\mathsf{FHEW}} N \sigma_{\mathsf{br}}}}}$.

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. 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.

E Discussion on Additional Parameters

E.1 Estimate Parameters for the Noise Flooding Technique

Here we give a rough parameter estimate for TFHE using noise flooding. We use our estimator for the DS-WM method setting the number of washing cycles to 1.

In fact the noise flooding method is a special case of the DS-WM method. In fact we will start by modifying the DS-WM-Int parameter set. Let's set the modulus to 2^{110} and ring to 2^{12}. Note that the ring is bigger that the rings that we used, but security is below 128-bit, according to the FHE-Standard [ACC+18]! So the parameter set doesn't satisfy out conditions, but let's make it easier. The blind rotation error is already above $B = 26$ bits, so we need at least $B * 2^{80} = 2^{106}$ bits of flooding noise according to the smudging lemma (Lemma 11). A 110-bits modulus should be large enough to accommodate the message. All other parameters (the decomposition base etc.) stay the same so that we don't need to increase the modulus anymore. Then we need $(14 + 1) \cdot 2$ convolutions per external product ($912 \cdot 30 = 27360$ total in a larger ring), a masking key of size 3977 MB (in comparison to 186 MB), a 1464 MB bootstrapping key (we have 134 MB), and 837 MB Key switching key (we have 79 MB). Quadruple precision FFT or Intel Hexl don't handle such big numbers. But we can use the RNS representation at the cost of roughly 2times more convolutions. In total, we get $27360 \cdot 2 = 54720$ convolutions, in comparison to 12768 convolutions in Our-Int parameter set. To summarize, naive flooding has 4.28 times more convolutions with a ring that is twice as large and requires 6278 MB of key material in comparison to our 399 MB. Remind that this parameter set doesn't give 128-bit security, but is close to. To satisfy our security constraint we would actually need to take a 2^{13} dimension ring! Note that the number of convolutions is just an indication of how much slower an implementation can be in best case. To implement RNS there is much more effort necessary, that is going to slow down computation due to composition, memory access, cache related issue etc.