Published in C. Cid and N. Yanai, Eds., *Advances in Information and Computer Security (IWSEC 2025)*, vol. 16208 of *Lecture Notes in Computer Science*, pp. 45-64, Springer, 2025

Approximate CRT-Based Gadget Decomposition for Fully Homomorphic Encryption

Olivier Bernard and Marc Joye

Zama, Paris, France
firstname.lastname@zama.ai

Abstract. Managing noise growth is a central challenge in fully homomorphic encryption (FHE). Gadget decomposition mitigates this by representing elements as vectors whose inner product with a gadget vector approximately reconstructs the original value. Radix-based decompositions support approximation but CRT-based ones have, so far, required exactness. We introduce, for the first time, CRT-based gadget decompositions *in the approximate setting*, combining the benefits of approximate decompositions with the structural advantages of CRT-based methods. This enables efficient blind rotation and (programmable) bootstrapping in TFHE using only native arithmetic while increasing parallelism. On a typical FPGA (17-bit multipliers), our approach achieves a speedup of over 2× and approximately 50% lower area than comparable radix-based approximate designs. The methodology also reduces bandwidth, memory, and compute in settings with large ciphertext moduli (e.g., 128-bit), benefiting both hardware and software implementations.

Keywords: Lattice-based cryptography · Fully homomorphic encryption (FHE) · Gadget decomposition · Blind rotation · Chinese remainder theorem (CRT) · Number-theoretic transform (NTT)

1 Introduction

Fully homomorphic encryption (in short, FHE) [30, 18] is often regarded as the holy grail of cryptography. Unlike traditional encryption methods, FHE allows direct computation on encrypted data, without requiring prior decryption. The result of the computation remains encrypted, ensuring end-to-end data security throughout the process. We refer the reader to [20, 8] for excellent surveys on fully homomorphic encryption.

Apart from a few exceptions, most known instantiations of FHE rely on lattice-based cryptography, basing their security on the learning with errors (LWE) problem [29] or its variants. As a result, the corresponding ciphertexts must be noisy to ensure security. While noise is generally not a concern in standard encryption schemes, it requires careful management in the context of fully homomorphic encryption. The main issue arises from the fact that noise within ciphertexts grows as they are processed homomorphically. If the noise exceeds a certain threshold, the ciphertext can no longer be decrypted correctly. There are

two primary strategies to address this challenge: *(i)* bootstrapping ciphertexts and *(ii)* controlling noise growth during computations.

The concept of *bootstrapping* was introduced in Gentry's seminal work in 2009 [17]. It consists in homomorphically evaluating the decryption circuit using an encrypted ciphertext and an encrypted decryption key as inputs. The result is a new ciphertext that encrypts the same plaintext—this process is also known as *recryption*. Since the decryption removes noise, the noise in a bootstrapped ciphertext is reset to a nominal level; i.e., the output ciphertext only contains the noise resulting from the bootstrapping process.

An alternative or complementary strategy for dealing with the noise is to ensure that the noise does not grow too quickly so that a larger number of homomorphic operations can be performed before bootstrapping becomes necessary. A common technique for this is *gadget decomposition* [26, 6]: when multiplying a noisy ciphertext by a scalar, the scalar is first decomposed with respect to a small radix B. Specifically, if Enc denotes a homomorphic encryption algorithm, the ciphertext $C \leftarrow \mathrm{Enc}(k \cdot x)$ is obtained by writing $k = \sum_{j=1}^{\ell} k_j \, B^{j-1}$ with $-\lfloor B/2 \rfloor \leq k_j \leq \lfloor B/2 \rfloor$ and then evaluating $\sum_{j=1}^{\ell} k_j \, \mathrm{Enc}(B^{j-1} \, x)$ from the ℓ ciphertexts $\mathrm{Enc}(x), \mathrm{Enc}(Bx), \ldots, \mathrm{Enc}(B^{\ell-1}x)$. The vector $(k_1, \ldots, k_\ell)$ is called the gadget decomposition of k. A quick analysis shows that, compared to the direct approach of getting $\mathrm{Enc}(k \cdot x)$ as $k \, \mathrm{Enc}(x)$, the noise better behaves using the gadget decomposition. Assuming that the noise in the input ciphertexts follows a Gaussian error distribution $\mathcal{N}(0, \sigma^2)$, the variance of the noise in the output ciphertexts $C \leftarrow k \, \mathrm{Enc}(x)$ and $C \leftarrow \sum_{j=1}^{\ell} k_j \, \mathrm{Enc}(B^{j-1} \, x)$ is respectively of $k^2\sigma$ and of $\left(\sum_{j=1}^{\ell} k_j^2\right)\sigma$—observe that as ℓ increases, $\sum_{j=1}^{\ell} k_j^2 \ll k^2$.

The gadget decomposition is not restricted to managing the noise in the scalar multiplication of ciphertexts, it is also central in the design of most FHE schemes as an auxiliary tool for certain FHE procedures; e.g., [6, 26, 5, 19, 1, 14, 15, 9, 11, 4]. Of special importance is the gadget decomposition when applied to improve bootstrapping procedures. In particular, similar to [1, 14], the bootstrapping in the TFHE scheme, building on [15], makes use of an accumulator that is updated in a for-loop according to encryptions of the secret key bits. This operation is referred to as *blind rotation* in [11]. It consists of a succession of external products which comprise polynomial multiplications and gadget decompositions. The technique equally applies to the programmable version of the bootstrapping [12]. On input an encryption of x, the output is an encryption of $f(x)$—with a nominal level of noise as it is the output of a bootstrapping procedure. The regular bootstrapping corresponds to function f being the identity map. A detailed description of the programmable bootstrapping with companion algorithms can be found in [22].

An essential ingredient to efficiency of TFHE and its variants is to perform only a radix-based gadget decomposition *up to a certain precision*; i.e., the least significant digits in the decomposition are dropped. This has two immediate benefits: *(i)* the performance of the (programmable) bootstrapping is greatly improved as each external product within the blind rotation involves ℓ-dimensional polynomial vectors and *(ii)* the overall size of the bootstrapping keys is signif-

icantly reduced as it is proportional to ℓ (namely, the number of digits in the radix-based gadget decomposition). Such an optimization seems however inherently limited to radix or mixed-radix based decompositions [21, Section 4.2].

Alternative gadget decompositions have been considered, including representations relying on the Chinese Remainder Theorem (CRT) [3, Section B.4]. Operations modulo the small factors can also be grouped in a two-level way, as demonstrated in [25]. This is mostly useful for large ciphertext moduli as in CKKS-like schemes [9]; see also [2] for an extension using a bivariate polynomials formalism.

Chinese remaindering is a natural method for handling large integers using small arithmetic chunks but it ought to be *exact*. Indeed, CRT-based gadget decomposition are extremely sensitive to errors, as these are getting spread by the inverse CRT isomorphism. It is therefore no longer possible to drop "digits" in the decomposition. This has unfortunate consequences both in terms of computational costs and of key sizes. In practice, that outweighs the benefits of using a CRT-based decomposition in the first place.

Our techniques and results: CRT-based gadget decomposition and approximate setting seem to be inherently incompatible. This work shows that this common belief is unfounded. We propose and develop methods for *approximate* gadget decompositions in a CRT-like manner. The proposed methods are generic and rely only on efficient arithmetic on "arithmetic-unit words." Being agnostic to the selected parameters, they smoothly fit with the various flavors of the number-theoretic transform (NTT) for polynomial multiplication.

As a concrete illustration, we demonstrate how plugging our approximate CRT-based gadget decompositions allows performing the whole *Blind Rotation* using only arithmetic modulo small moduli. An application to the programmable bootstrapping of TFHE-like ciphertexts leads to a number of significant advantages:

1. All arithmetic units can work completely independently in parallel, provided they synchronize for the gadget decomposition itself, but only for this step.
2. The NTT/iNTT transforms modulo the ciphertext modulus q are replaced by several transforms modulo smaller moduli, ideally that fit into a single machine word. This is interesting since *(i)* the computational complexity of these NTT/iNTT transforms also depends on $\mathsf{M}(q)$, i.e., on the word size of q, and *(ii)* there is no need to lift everything up modulo q.
3. Including the twisting factors in the CRT encodings of the bootstrapping keys and test polynomial (used to program the bootstrapping) further simplifies the computation of the gadget decomposition itself, hence incurring minimal cost.

Furthermore, in addition to important complexity benefits/improvements, the resulting implementation also saves in both bandwidth and storage. In particular, being in the approximate setting, the bootstrapping keys are much more compact.

2 Preliminaries

Throughout the paper, elements in $\mathbb{Z}/q\mathbb{Z}$, the ring of integers modulo q, are viewed as integers in the range $[\![-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor]\!]$, where $\lfloor \cdot \rfloor$ denotes the flooring function. When integers modulo q are seen as integers, or more precisely by their integer representatives, this is indicated by the lifting function; for an integer $a \in \mathbb{Z}/q\mathbb{Z}$, this is written as $\big(a \mod q\big)_{\mathbb{Z}}$ or sometimes, more simply, as $\big(a\big)_{\mathbb{Z}}$. Vectors are given in row representation and denoted by bold letters $\boldsymbol{v}$. Polynomials, as well as algebraic integers, are denoted by cursive letters $\mathscr{a}$. If $\mathcal{S}$ is a set, $a \xleftarrow{\$} \mathcal{S}$ indicates that a is sampled uniformly at random in $\mathcal{S}$. If χ is a probability distribution, $a \leftarrow \chi$ indicates that a is sampled according to χ.

2.1 Gadget Decomposition

Gadgets decompose elements as vectors of small pieces whose inner product with a so-called *gadget vector* reconstructs (an approximation of) the original elements. In the FHE context, these gadget decompositions allow controlling the noise growth e.g., for the multiplication of a ciphertext by a scalar. The gadget decomposition is called *exact* when the recomposing retrieves completely the original element. As aforementioned, one important characteristic of the TFHE scheme is to rely on an *approximate* gadget decomposition [10, 7], where only an approximation of the original element is retrieved. This results in smaller bootstrapping keys and improved bootstrapping performance.

We give here a formal generic definition to gadget-decompose elements. For the sake of clarity, we address the case of number field elements, which covers most instantiations of FHE schemes. It is useful to introduce some notation. A number field $\mathcal{K}$ is a finite extension of the field $\mathbb{Q}$ of rational numbers. The ring of integers $\mathcal{R}$ of $\mathcal{K}$ is the set of all algebraic integers contained in $\mathcal{K}$. For an integer q, the residue ring $\mathcal{R}/q\mathcal{R}$ of $\mathcal{R}$ modulo q is denoted $\mathcal{R}_q$. This general setting encompasses two important sub-cases for FHE applications:

- $\mathcal{K} = \mathbb{Q}$, in which case $\mathcal{R} = \mathbb{Z}$ and $\mathcal{R}_q = \mathbb{Z}/q\mathbb{Z}$;
- $\mathcal{K} = \mathbb{Q}(\zeta_m) \cong \mathbb{Q}[x]/\langle \Phi_m(x) \rangle$, in which case $\mathcal{R} = \mathbb{Z}[\zeta_m] \cong \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$ and $\mathcal{R}_q = (\mathbb{Z}/q\mathbb{Z})[\zeta_m] \cong (\mathbb{Z}/q\mathbb{Z})[x]/\langle \Phi_m(x) \rangle$ where ζ_m is any primitive m-th root of unity (e.g., $\zeta_m = \exp(2\pi i/m)$) and Φ_m is the m-th cyclotomic polynomial.

Definition 1 (Adapted from [11, Definition 3.6]). *Using the previous notations, a* gadget decomposition on $\mathcal{R}_q$ *of level ℓ, quality β, and precision ε is given by:*

1. *a gadget vector* $\mathbf{g} = (\mathscr{g}_1, \ldots, \mathscr{g}_\ell) \in \mathcal{R}_q^\ell$;
2. *an efficient algorithm* $\nabla := \nabla_{\mathbf{g}}^{\beta,\varepsilon} \colon \mathcal{R}_q \to \mathcal{R}^\ell$ *such that for any* $\mathscr{a} \in \mathcal{R}_q$*:*

$$\|\nabla \mathscr{a}\|_\infty \leq \beta \quad \textit{and} \quad \|\mathscr{a} - \langle \nabla \mathscr{a}, \mathbf{g} \rangle\|_\infty \leq \varepsilon \,,$$

where the infinity norms are always taken component-wise.

Gadget sub- or super-scripts are generally omitted for readability.

The definition naturally extends to other mathematical structures like the real discretized torus $\mathbb{T}_q \coloneqq \frac{1}{q}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T} \coloneqq \mathbb{R}/\mathbb{Z}$ by identifying $\mathbb{T}_q$ with $\mathbb{Z}/q\mathbb{Z}$ or, more generally, like its polynomial variant $\mathbb{T}_q[x]/\langle \Phi_m(x) \rangle$ by identifying it with $(\mathbb{Z}/q\mathbb{Z})[x]/\langle \Phi_m(x) \rangle$; cf. [22, Remark 3]. Alternatively, the gadget algorithm with parameters $(\ell, \beta, \varepsilon)$ can be directly defined as $\nabla_{\mathbf{g}}^{\beta,\varepsilon} \colon \mathbb{T}_q[x]/\langle \Phi_m(x) \rangle \to \big(\mathbb{Z}[x]/\langle \Phi_m(x) \rangle\big)^\ell$ for some gadget vector $\mathbf{g} \in \big(\mathbb{T}_q[x]/\langle \Phi_m(x) \rangle\big)^\ell$, viewing the set $\mathbb{T}[x]/\langle \Phi_m(x) \rangle$ as a $\mathbb{Z}[x]/\langle \Phi_m(x) \rangle$-module.

Radix-based gadget decomposition: Let q be a modulus such that B^ℓ divides q for some integers $B > 1$ and $1 \le \ell \le \lfloor \log_B q \rfloor$. A radix-based gadget decomposition of quality β and level ℓ is given by the gadget vector $\mathbf{g} = \big(\frac{q}{B}, \ldots, \frac{q}{B^\ell}\big)$.

For any $a \in \mathbb{Z}/q\mathbb{Z}$, the decomposition algorithm returns the ℓ most significant digits of a in radix B, where a is viewed as an integer in $[\![-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor]\!]$. Each digit is selected so that its amplitude is bounded by $\beta = \lfloor \frac{B}{2} \rfloor$; specifically, we write $a \equiv \sum_{j=1}^{\ell} a_j \frac{q}{B^j} + R \pmod q$ with $-\lfloor B/2 \rfloor \le a_j \le -\lfloor B/2 \rfloor$ and $|R| < q/(2B^\ell)$. Such a decomposition is always possible. Letting $\nabla a = (a_1, \ldots, a_\ell)$, the corresponding precision is then of $\varepsilon = \lfloor \frac{q}{2B^\ell} \rfloor$. Indeed, we have $a - \langle \nabla \mathscr{a}, \mathbf{g} \rangle \equiv a - \sum_{j=1}^{\ell} a_j \frac{q}{B^j} \equiv R \pmod q$ and $|R| \le \lfloor q/(2B^\ell) \rfloor$. It is worth remarking that $\varepsilon = 0$ when $q = B^\ell$.

The radix-B gadget decomposition extends to $\mathcal{R}_q$ by applying ∇ to each coefficient of a polynomial $\mathscr{a} \in \mathcal{R}_q$; in this particular case, the components of the above $\mathbf{g}$ are simply embedded in $\mathcal{R}_q$, i.e., as scalars in $\mathbb{Z}/q\mathbb{Z} \subset \mathcal{R}_q$, but in general those could be any $\mathscr{g}_j \in \mathcal{R}_q$.

Mixed-radix gadget decompositions generalize radix-B decompositions to modulus q such that $Q \coloneqq \prod_{j=1}^{\ell} q_j$ divides q, for (non-necessarily distinct) factors q_j. The gadget vector is defined as $\mathbf{g} = \big(\frac{q}{q_1}, \frac{q}{q_1 q_2} \ldots, \frac{q}{q_1 q_2 \cdots q_\ell}\big)$. The quality is of $\beta = \lfloor \max_j q_j/2 \rfloor$ and the precision is of $\varepsilon = \left\lfloor \frac{q}{2Q} \right\rfloor$. Radix-$B$ gadget decompositions correspond to the special case $q_1 = q_2 = \cdots = q_\ell = B$.

CRT-based gadget decomposition: Instead of the radix-B representation, the CRT-based gadget decomposition considers the Chinese Remainder Theorem (CRT) isomorphism as the decomposition algorithm. Let $q_1, \ldots, q_\ell$ be pairwise co-prime integers and let $q = \prod q_j$. The gadget vector is defined as

$$\mathbf{z} = (z_1, \ldots, z_\ell) \quad \text{where } z_j = \tilde{q}_j \cdot \big(\tilde{q}_j^{-1} \mod q_j\big)_{\mathbb{Z}}$$

for $\tilde{q}_j = \prod_{\substack{1 \le k \le \ell \\ k \ne j}} q_k$.

The CRT maps any element $a \in \mathbb{Z}/q\mathbb{Z}$ to

$$\nabla_{\mathbf{z}} a \coloneqq (\underbrace{a \mod q_1}_{=a_1}, \ldots, \underbrace{a \mod q_\ell}_{=a_\ell}) \,,$$

and the inverse isomorphism is explicitly written as the following inner product modulo q:

$$a \equiv \langle \nabla_{\mathbf{z}} a, \mathbf{z} \rangle \equiv \left(\sum_{j=1}^{\ell} a_j \cdot z_j \right) \pmod{q} \ .$$

The correctness is easily verified by checking that $z_j \equiv 1 \pmod{q_j}$ and that for $k \neq j$, $z_k \equiv 0 \pmod{q_j}$.

Therefore, for the CRT-based gadget decomposition, the gadget vector is $\mathbf{z}$ as defined above, and the decomposition algorithm $\nabla_{\mathbf{z}}$ simply consists in the ℓ modulo operations. This yields an *exact* ($\varepsilon = 0$) gadget decomposition on $\mathbb{Z}/q\mathbb{Z}$ of level ℓ and quality $\beta = \max_i \left\lfloor \frac{q_i}{2} \right\rfloor$.

By nature, the CRT-based decomposition is intrinsically incompatible with approximate decompositions. Indeed, dropping any CRT "digit" results in a big error of order $z_i \approx q/\beta$.

The CRT-based gadget decomposition readily extends to $\mathcal{R}_q$. Consider an algebraic integer $\mathfrak{f} \in \mathcal{R}_q$ written as the polynomial $\mathfrak{f} = \sum_{i=0}^{N-1} f_i \, x^i$ with $f_i \in \mathbb{Z}/q\mathbb{Z}$. Each polynomial coefficient of $\mathfrak{f}$ is replaced with

$$f_i \longmapsto \nabla_{\mathbf{z}} f_i \coloneqq \big(\underbrace{f_i \mod q_1}_{=f_{i,1}}, \ldots, \underbrace{f_i \mod q_\ell}_{=f_{i,\ell}} \big)$$

and the ℓ polynomials

$$\begin{cases} \mathfrak{f}_1 = \mathfrak{f} \mod q_1 = \sum_{i=0}^{N-1} f_{i,1} \, x^i \\ \quad \vdots \\ \mathfrak{f}_\ell = \mathfrak{f} \mod q_\ell = \sum_{i=0}^{N-1} f_{i,\ell} \, x^i \end{cases} .$$

are formed. The vector $\nabla_{\mathbf{z}} \mathfrak{f} = \big(\mathfrak{f}_1, \ldots, \mathfrak{f}_\ell \big) \in \mathcal{R}^\ell$ represents the CRT-based gadget decomposition of $\mathfrak{f}$. The corresponding gadget vector $\mathbf{z} \in \mathcal{R}_q^\ell$ is defined with the same coefficients z_i as in the integer case, but now viewed as constant polynomials in $\mathcal{R}_q$. It is easy to verify that $\mathfrak{f} - \langle \nabla_{\mathbf{z}} \mathfrak{f}, \mathbf{z} \rangle \equiv 0 \pmod{q}$, and thus $\varepsilon = 0$. Further, if $\beta = \max_i \left\lfloor \frac{q_i}{2} \right\rfloor$ then $\| \nabla_{\mathbf{z}} \mathfrak{f} \|_\infty \leq \beta$, where the infinity norm of a polynomial is defined as the infinity norm of the vector of its coefficients.

2.2 Fully Homomorphic Encryption

Generalized LWE samples: Let $\mathcal{R}$ denote the ring of integers of some number field and let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Let also χ denote some error distribution over $\mathcal{R}$. Given a private vector $\mathfrak{s} \in \mathcal{R}^k$, a *generalized LWE sample* is a vector of the form

$$\big(\mathfrak{a} = (a_1, \ldots, a_k), \mathfrak{r} \big) \in \mathcal{R}_q^{k+1} \quad \text{where } \mathfrak{r} = \langle \mathfrak{a}, \mathfrak{s} \rangle + \mathfrak{e}$$

with $\mathfrak{a} \xleftarrow{\$} \mathcal{R}_q^k$ and $\mathfrak{e} \leftarrow \chi$. Given a fresh sample $(\mathfrak{a}, \mathfrak{r}) \in \mathcal{R}_q^{k+1}$, (the encoding of) a message μ in $\mathcal{R}_q$, called plaintext, is encrypted under key $\mathfrak{s}$ to form the ciphertext

$$\mathfrak{C} \leftarrow \mathsf{GLWE}_{\mathfrak{s}}(\mu) \coloneqq (\mathfrak{a}, \mathfrak{r} + \mu) \in \mathcal{R}_q^{k+1} \ .$$

Two specialized instances are used:

1. $\mathcal{R}_q \cong (\mathbb{Z}/q\mathbb{Z})[x]/\langle x^N + 1\rangle$ with N a power of 2 and $k = 1$: this is referred to as the Ring-LWE (or RLWE) assumption;
2. $\mathcal{R}_q = \mathbb{Z}/q\mathbb{Z}$ and $k > 1$: this is the original LWE assumption.

The matching samples are respectively called LWE samples and RLWE samples.

Related homomorphic operations: Once a gadget decomposition $\nabla := \nabla_{\mathbf{g}}^{\beta,\varepsilon}$ has been fixed relatively to some gadget vector $\mathbf{g} = (g_1, \ldots, g_\ell) \in \mathcal{R}_q^\ell$, it induces an associated *leveled* encryption of a message $m \in \mathcal{R}$, as

$$\mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m) = \big(\mathsf{GLWE}_{\mathfrak{z}}(g_j \cdot m)\big)_{1\le j\le \ell} ,$$

and its GGSW expansion

$$\mathsf{GGSW}_{\mathfrak{z}}(m) = \big(\mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(-\mathfrak{z}_1 \cdot m), \ldots, \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(-\mathfrak{z}_k \cdot m), \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m)\big) .$$

When $k = 1$, the associated leveled encryption and corresponding expansion are respectively written $\mathsf{RLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m)$ and $\mathsf{RGSW}_{\mathfrak{z}}(m)$. Leveled encryptions are sometimes denoted with a prime ($'$) e.g., as in [27] or with the Lev suffix e.g., as in [13]; above notation is preferred as it makes more apparent the underlying gadget decomposition.

Following [27], this allows defining certain homomorphic operations. These operations do not depend, formula-wise, on the particular gadget decomposition. Only their noise analysis may differ, depending on ℓ, β, ε and on the distribution of $\nabla(\cdot)$.

Scalar product: The gadget decomposition gives rise to the definition of a scalar product:

$$\odot\colon \mathcal{R}_q \times \mathcal{R}_q^\ell \to \mathcal{R}_q, \big(\ell, \boldsymbol{\hbar}\big) \mapsto \ell \odot \boldsymbol{\hbar} := \langle \nabla_{\mathbf{g}}\ell, \boldsymbol{\hbar}\rangle .$$

In particular, if the polynomial vector $\boldsymbol{\hbar}$ is the gadget vector, we have $\ell \odot \mathbf{g} \approx \ell$.

Typically, this is extended to compute the product of a known element $\alpha \in \mathcal{R}_q$ with an encryption of a message m to get an encryption of $\alpha \cdot m$. Letting $\nabla\alpha = (\alpha_1, \ldots, \alpha_\ell)$, it can be seen that

$$\alpha \odot \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m) := \big\langle \nabla\alpha, \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m)\big\rangle = \mathsf{GLWE}_{\mathfrak{z}}(\alpha \cdot m) .$$

By evaluating $\big((\alpha \cdot g_j) \odot \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m)\big)_{1\le j\le \ell}$, one so gets $\mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(\alpha \cdot m)$ as an output.

External product: The external product allows computing the GLWE encryption of the product of two encrypted messages, as

$$\begin{aligned}
&\mathsf{GLWE}_{\mathfrak{z}}(\mu_1) \circledast \mathsf{GGSW}_{\mathfrak{z}}(m_2) \\
&\qquad := \Big(\textstyle\sum_{j=1}^{k} a_j \odot \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(-\mathfrak{z}_j \cdot m_2)\Big) + b \odot \mathsf{GLWE}_{\mathfrak{z}}^{\nabla,\mathbf{g}}(m_2) \\
&\qquad = \mathsf{GLWE}_{\mathfrak{z}}(\mu_1 \cdot m_2 + e \cdot m_2)
\end{aligned}$$

where $\big(\mathcal{a}_1, \ldots, \mathcal{a}_k, \mathcal{b} = \sum_{j=1}^{k} \mathcal{a}_j \cdot \mathcal{s}_j + \mu_1 + \mathcal{e}\big)$ expands the input $\mathsf{GLWE}_{\mathcal{s}}(\mu_1)$. The result is a GLWE encryption of $\mu_1 \cdot \mathcal{m}_2$ if message $\mathcal{m}_2$ is small so that $\|\mathcal{e} \cdot \mathcal{m}_2\|_\infty \approx \|\mathcal{e}\|_\infty$. The external product is asymmetric in the sense that one of its operand is a GLWE ciphertext whereas the other is a GGSW ciphertext with $(k+1)\ell$ components.

3 An Approximate CRT-Based Gadget Decomposition

In this section, we propose to realize an approximate CRT-based gadget decomposition *via* a decomposition which is half-way between CRT-based and mixed-radix-based gadget decompositions. It relies on a two-congruence Chinese Remainder Algorithm, as described in [28, Section 2.2], and on a classical CRT decomposition. At high level, the modulus is decomposed into a high part and a low part, which serve as a basis for the mixed-radix decomposition, wherein the low part will be dropped. The low and high parts are further decomposed using the CRT representation. Intuitively, the size of the low part controls the precision ε of the decomposition, whilst the size of the CRT moduli controls its quality β.

3.1 Motivation

Using the CRT gadget decomposition outlined in Section 2.1, any $\mathcal{f} \in \mathcal{R}_q$ may be expressed *exactly* as $\mathcal{f} = \langle \nabla_{\mathbf{z}} \mathcal{f}, \mathbf{z} \rangle \mod q$. However, some applications only require an approximate expression $\tilde{\mathcal{f}}$ for $\mathcal{f}$, provided that $\tilde{\mathcal{f}}$ satisfies $\|\mathcal{f} - \tilde{\mathcal{f}}\|_\infty \leq \varepsilon$ for some given bound ε. One such example is when a ciphertext is gadget-decomposed. The lower part contains noise; a full gadget decomposition boils down at some point to uselessly decompose noise. We illustrate this in the case of LWE ciphertexts for simplicity but the same carries over e.g., RLWE ciphertexts or other types of ciphertexts. Consider an LWE-type ciphertext $\boldsymbol{C} = \big(\boldsymbol{a} = (a_1, \ldots, a_n), b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \mu + e\big) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ where $\mu = \lfloor q/t \rfloor m$ encodes a message $m \in \mathbb{Z}/t\mathbb{Z}$, $\boldsymbol{s} \in \{0,1\}^n$ is the secret key, and noise $e \in \mathbb{Z}$ is sampled according to Gaussian distribution $\mathcal{N}(0, \sigma^2)$. The phase and error functions of $\boldsymbol{C}$ are respectively defined by $\varphi_{\boldsymbol{s}}(\boldsymbol{C}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle \mod q$ and $\mathrm{Err}(\boldsymbol{C}) = \big(\varphi_{\boldsymbol{s}}(\boldsymbol{C}) - \mu\big)_{\mathbb{Z}}$.

Let $\tilde{\boldsymbol{C}} \coloneqq \langle \nabla_{\mathbf{g}} \boldsymbol{C}, \mathbf{g} \rangle \mod q = (\tilde{\boldsymbol{a}}, \tilde{b})$. Noting that

$$\begin{aligned} \varphi_{\boldsymbol{s}}(\tilde{\boldsymbol{C}}) &\equiv \varphi_{\boldsymbol{s}}(\tilde{\boldsymbol{C}} - \boldsymbol{C}) + \varphi_{\boldsymbol{s}}(\boldsymbol{C}) \equiv \tilde{b} - b - \langle \tilde{\boldsymbol{a}} - \boldsymbol{a}, \boldsymbol{s} \rangle + \varphi_{\boldsymbol{s}}(\boldsymbol{C}) \\ &\equiv \mathfrak{b} - \langle \mathfrak{a}, \boldsymbol{s} \rangle + \varphi_{\boldsymbol{s}}(\boldsymbol{C}) \pmod q \end{aligned}$$

for some variables $\mathfrak{a} \in [\![-\varepsilon, \varepsilon]\!]^n$ and $\mathfrak{b} \in [\![-\varepsilon, \varepsilon]\!]$ and assuming that $\mathfrak{a}$ and $\mathfrak{b}$ are uniformly distributed, the variance of the noise error in the recomposed ciphertext $\tilde{\boldsymbol{C}}$ verifies

$$\begin{aligned} \mathrm{Var}\big(\mathrm{Err}(\tilde{\boldsymbol{C}})\big) &= \mathrm{Var}\left(\big(\varphi_{\boldsymbol{s}}(\tilde{\boldsymbol{C}} - \mu)\big)_{\mathbb{Z}}\right) = \mathrm{Var}\big(\mathfrak{b} - \langle \mathfrak{a}, \boldsymbol{s} \rangle\big) + \mathrm{Var}\big(\mathrm{Err}(\boldsymbol{C})\big) \\ &= \mathrm{Var}(\mathfrak{b}) + n\big(\mathrm{Var}(\mathfrak{a}_j)\,\mathrm{Var}(s_j) + \mathrm{Var}(\mathfrak{a}_j)\mathbb{E}[s_j]^2 \\ &\qquad + \mathrm{Var}(s_j)\mathbb{E}[\mathfrak{a}_j]^2\big) + \sigma^2 \\ &= \tfrac{1}{6}(n+2)\varepsilon(\varepsilon+1) + \sigma^2 \leq \tfrac{n+2}{3}\,\varepsilon^2 + \sigma^2 \end{aligned}$$

since $\mathrm{Var}(\mathfrak{b}) = \mathrm{Var}(\mathfrak{a}_j) = \frac{1}{12}\big((2\varepsilon+1)^2-1\big) = \frac{1}{3}\varepsilon(\varepsilon+1)$, $\mathrm{Var}(s_j) = \frac{1}{4}$, $\mathbb{E}[s_j] = \frac{1}{2}$, and $\mathbb{E}[\mathfrak{a}_j] = 0$.

As a result, if the bound ε on the approximation error $\|\tilde{\boldsymbol{C}} - \boldsymbol{C}\|_\infty$ is for example set such that $\varepsilon \leq \sigma\sqrt{3/(n+2)}$ then $\mathrm{Var}\big(\mathrm{Err}(\tilde{\boldsymbol{C}})\big) \leq 2\sigma^2$; i.e., the impact on the noise error is very low. Regarding the performance, the impact can however be substantial as will be apparent in Section 4.

3.2 Description

Formally, let $q = Q \cdot Q_{\mathsf{low}}$ with $\gcd\big(Q, Q_{\mathsf{low}}\big) = 1$, where the high part $Q = \prod_{j=1}^{\ell} q_j$ (resp. low part $Q_{\mathsf{low}} = \prod_{j=1}^{k} q'_j$) is a product of ℓ (resp. k) pairwise co-prime integers $q_1, \ldots, q_\ell$ (resp. $q'_1, \ldots, q'_k$).

The definition of the gadget vector for our approximate CRT-based gadget decomposition is similar to what it would be for an *exact* CRT reconstruction, but omitting the coefficients corresponding to the divisors of Q_{low}, i.e.,

$$\mathbf{w} = \big(w_1, \ldots, w_\ell\big) \in \mathcal{R}_q^\ell \,,$$

where

$$w_j = Q_{\mathsf{low}}\widetilde{Q}_j \cdot \Big(\big(Q_{\mathsf{low}}\widetilde{Q}_j\big)^{-1} \bmod q_j\Big)_{\mathbb{Z}} \quad \text{and} \quad \widetilde{Q}_j = \frac{Q}{q_j} \,. \tag{1}$$

The approximate CRT-based gadget decomposition of $f = \sum_{i=0}^{N-1} f_i\, x^i \in \mathcal{R}_q$ is then given by the ℓ-tuple

$$\nabla_{\mathbf{w}} f = \big(f_1, \ldots, f_\ell\big) \in \mathcal{R}^\ell \,,$$

where, for $1 \leq j \leq \ell$, $f_j = \sum_{i=0}^{N-1} f_{ij}\, x^i$ for $f_{ij} \in \mathbb{Z}/q_j\mathbb{Z}$ defined by the congruence

$$f_{ij} \equiv f_i - \sum_{u=1}^{k} \tfrac{Q_{\mathsf{low}}}{q'_u} \cdot \Big(\big(\tfrac{Q_{\mathsf{low}}}{q'_u}\big)^{-1} \cdot f_i \mod q'_u\Big)_{\mathbb{Z}} \mod q_j \,. \tag{2}$$

We stress that computing $\nabla_{\mathbf{w}}$ never involves arithmetic operations modulo integers bigger than the chosen divisors of Q_{low} and Q. Indeed, the sum indexed by u in Equation (2) has no dependency in j: for all divisors q'_u of Q_{low}, the part modulo q'_u of each term can be computed beforehand by units working solely modulo q'_u. Once these values are disclosed to units working modulo divisors q_j of Q, the products with the precomputed twisting terms $\Big\{\frac{Q_{\mathsf{low}}}{q'_1} \mod q_j, \ldots, \frac{Q_{\mathsf{low}}}{q'_k} \mod q_j\Big\}$ can be directly performed modulo q_j.

Proposition 1. *The gadget vector* $\mathbf{w}$ *given by Equation (1) and the associated decomposition algorithm* $\nabla_{\mathbf{w}}$ *given by Equation (2), define a level-*ℓ *gadget decomposition on* $\mathcal{R}_q$ *of quality and precision given by the following bounds, for all* $f \in \mathcal{R}_q$*:*

$$\|\nabla_{\mathbf{w}} f\|_\infty \leq \beta = \max_{1\leq j\leq \ell} \big\lfloor \tfrac{q_j}{2} \big\rfloor \quad \textit{and} \quad \|f - \langle \nabla_{\mathbf{w}} f, \mathbf{w}\rangle\|_\infty \leq \varepsilon = k \cdot \big\lfloor \tfrac{Q_{\mathsf{low}}}{2} \big\rfloor \,,$$

where the infinity norms are understood coefficient-wise.

Remark 1. Recall that the congruence classes f_{ij}'s are typically represented as integers in $[\![-\lfloor \frac{q_j}{2} \rfloor, \lfloor \frac{q_j}{2} \rfloor]\!]$. Any reasonable choice of representatives is also possible, in which case the bounds given in Proposition 1 might be slightly worse.

Proof (of Proposition 1). The only non-immediate statement is relative to the precision of the gadget decomposition. Let $\tilde{\ell} \coloneqq \langle \nabla_{\mathbf{w}} \ell, \mathbf{w} \rangle = \sum_{j=1}^{\ell} w_j \cdot \ell_j \mod q$. Extracting the Q_{low} factor from the w_j's yields that $\tilde{\ell}$ can be written as $Q_{\mathsf{low}} \cdot \big(\mathcal{F} \bmod Q\big)_{\mathbb{Z}}$, where

$$\mathcal{F} \coloneqq \sum_{j=1}^{\ell} \widetilde{Q}_j \cdot \left(\frac{\ell_j}{Q_{\mathsf{low}}} \cdot \widetilde{Q}_j^{-1} \bmod q_j \right)_{\mathbb{Z}} \mod Q \ .$$

Thus, by the CRT applied to the high part using $\gcd\big(Q_{\mathsf{low}}, Q\big) = 1$, for all $j \in [\![1, \ell]\!]$ we have that $\mathcal{F} \equiv \frac{\ell_j}{Q_{\mathsf{low}}} \pmod{q_j}$. Now, let $\mathcal{S}$ be the polynomial whose coefficients are given by the inner sum indexed by u in Equation (2), i.e., $\mathcal{S} \coloneqq \sum_{u=1}^{k} \widetilde{Q}'_u \cdot \left(\big(\widetilde{Q}'_u\big)^{-1} \cdot \ell \mod q'_u \right)_{\mathbb{Z}}$, where $\widetilde{Q}'_u = \frac{Q_{\mathsf{low}}}{q'_u}$ for $u \in [\![1, k]\!]$, so that $\ell_j \equiv \ell - \mathcal{S} \pmod{q_j}$ for all $j \in [\![1, \ell]\!]$. The first key observation about $\mathcal{S}$ is that, by the CRT applied to divisors of Q_{low}, $\mathcal{S} \equiv \ell \pmod{Q_{\mathsf{low}}}$. Hence, $\big(\ell - \mathcal{S}\big)$ is actually divisible by Q_{low}, which in turn implies $\tilde{\ell} = Q_{\mathsf{low}} \cdot \big(\mathcal{F} \mod Q\big)_{\mathbb{Z}} = Q_{\mathsf{low}} \cdot \left(\frac{\ell - \mathcal{S}}{Q_{\mathsf{low}}} \mod Q \right)_{\mathbb{Z}} = \ell - \mathcal{S} \mod q$. The second key observation about $\mathcal{S}$ is that its coefficients have amplitude bounded by $\|\mathcal{S}\|_\infty = \|\ell - \tilde{\ell}\|_\infty \leq k \cdot \left\lfloor \frac{Q_{\mathsf{low}}}{2} \right\rfloor$, yielding the result. □

Remark 2. In order to give more intuition about the proof, it seems interesting to mention that $\mathcal{S}$ is "almost" equal to $\big(\ell \bmod Q_{\mathsf{low}}\big)$. In fact, $\mathcal{S}$ *is* congruent to $\big(\ell \bmod Q_{\mathsf{low}}\big)$ by the CRT, but the reduction step modulo Q_{low} would not be computable directly modulo another q_j and would therefore involve arithmetic modulo Q_{low}. Skipping this reduction modulo Q_{low} is precisely what induces an approximation error which scales linearly in k.

4 Application to the Blind Rotation

The blind rotation is the costliest part of the (programmable) bootstrapping phase of TFHE-like schemes. Starting from a noisy LWE ciphertext, it consists in essence in applying iteratively an encrypted CMux operation on an accumulator, controlled by extended encryptions of the components of the initial LWE key, which constitute the bootstrapping keys.

In this section, we specialize it to the case where LWE keys are binary and to $2N$-th cyclotomic rings of the form $\mathcal{R} \cong \mathbb{Z}[x]/\langle x^N + 1 \rangle$. As the gadget decomposition is a low-level primitive, our new approximate CRT-based gadget decomposition also applies to broader settings, as other key distributions [23], e.g., ternary, other rings $\mathcal{R}$ [24], e.g., m-th cyclotomic rings where m is a prime or is of the form $2^a \cdot 3^b$, or $\mathcal{R}$-modules of rank greater than 1.

4.1 GINX Blind Rotation

Let q be the ciphertext modulus, let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} \cong (\mathbb{Z}/q\mathbb{Z})[x]/\langle x^N + 1\rangle$ be the $2N$-th cyclotomic ring modulo q and let t be the plaintext modulus. The *Blind Rotation* starts from an LWE encryption of dimension n of an encoding $\mu \in \mathbb{Z}/2N\mathbb{Z}$ of a message $m \in \mathbb{Z}/t\mathbb{Z}$, i.e., from

$$\mathsf{LWE}_{\boldsymbol{s}}(\mu) = \big(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s}\rangle + \mu + e\big) \in (\mathbb{Z}/2N\mathbb{Z})^{n+1} \ ,$$

where the noise e follows a sufficiently large Gaussian distribution and the key $\boldsymbol{s}$ is supposed to be binary, i.e., $\boldsymbol{s} = (s_1, \ldots, s_n) \in \{0,1\}^n$. In particular, we consider that the *Modulus Switching* from q to $2N$ has previously been done.

Bootstrapping keys: Suppose a gadget decomposition $\nabla := \nabla_{\mathbf{g}}$ of level ℓ has been fixed relatively to a gadget vector $\mathbf{g} = \big(g_1, \ldots, g_\ell\big) \in \mathcal{R}_q^\ell$. The encrypted CMux operations are enabled by RGSW encryptions associated to $\mathbf{g}$ of the bits of $\boldsymbol{s}$ under a key $\mathfrak{z} \in \mathcal{R}_q$. More precisely, the *bootstrapping keys* associated to $\mathbf{g}$ are hence defined, for $i \in [\![1, n]\!]$, by

$$\mathsf{bsk}[i] = \mathsf{RGSW}_{\mathfrak{z}}(s_i) = \left(\Big(\mathsf{RLWE}_{\mathfrak{z}}\big(g_j \cdot (-\mathfrak{z} \cdot s_i)\big)\Big)_{1\le j\le \ell}, \big(\mathsf{RLWE}_{\mathfrak{z}}(g_j \cdot s_i)\big)_{1\le j\le \ell}\right) \ .$$

We let $\mathsf{bsk}[i]_1$ (resp. $\mathsf{bsk}[i]_2$) denote the leveled encryption of $-\mathfrak{z}s_i$ (resp. s_i), i.e., the first (resp. second) part of $\mathsf{bsk}[i]$. Further, each leveled part is also indexed by j, so that e.g., $\mathsf{bsk}[i]_{2,j}$ refers to $\mathsf{RLWE}_{\mathfrak{z}}(g_j \cdot s_i)$.

Test polynomial: A so-called *test polynomial* enables the programmability in GINX bootstrapping. Suppose for simplicity that t is even and that function $f\colon \mathbb{Z}/t\mathbb{Z} \to \mathbb{Z}/t\mathbb{Z}$ is negacyclic; i.e., $f(x) = -f\big(x + \frac{t}{2}\big)$. The test polynomial can be then defined as

$$\mathfrak{v} = \left\lfloor \tfrac{q}{t} \right\rfloor \cdot \sum_{i=0}^{N-1} f\Big(\Big\lfloor i \cdot \tfrac{t}{2N} \Big\rceil\Big) \cdot x^i \quad \in \mathcal{R}_q \ .$$

For our purpose, it is sufficient to know that a suitable $\mathfrak{v} \in \mathcal{R}_q$ encoding f is given and that the *Blind Rotation* eventually computes an RLWE encryption of $\mathfrak{v} \cdot x^{-\mu - e}$, with nominal noise, from the LWE encryption of an encoding of m. In particular, if e is not too large, the constant coefficient of the output contains an encryption of an encoding of $f(m)$.

Encrypted CMuxes: The core operation in the loop of the *Blind Rotation* is the encrypted CMux gate, which starts from a RLWE encryption $\mathscr{C}$ of some $\boldsymbol{m}$ and outputs a RLWE encryption $\mathscr{C}'$ of $x^{s_i a_i} \cdot \boldsymbol{m}$. Concretely, this is achieved by computing

$$\mathscr{C}' \leftarrow \mathscr{C} + \Big(\big(x^{a_i} - 1\big) \cdot \mathscr{C}\Big) \circledast \mathsf{RGSW}_{\mathfrak{z}}\big(s_i\big) \ ,$$

noting that $x^{s_i a_i} \cdot \boldsymbol{m}$ is equal to $\boldsymbol{m}$ if $s_i = 0$, and to $x^{a_i} \cdot \boldsymbol{m}$ if $s_i = 1$.

This works in particular because the multiplication of $\mathscr{C}$ by x^{a_i} is actually a negacyclic permutation of the coefficients of $\mathscr{C}$ that does not induce any noise growth.

Computing the Blind Rotation loop: At very high level, the *Blind Rotation* starts from a trivial noiseless RLWE encryption $\mathtt{Acc} = \big(0, v \cdot x^{-b}\big) \in \mathcal{R}_q^2$, and then sequentially applies n times the above-defined CMux gate, as depicted in Algorithm 1.

Algorithm 1 GINX Blind Rotation with binary keys (high level)

Require: $\mathsf{LWE}_{\boldsymbol{s}}(\mu) = \big(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \mu + e\big)$, bootstrapping keys $\mathsf{bsk}[1 \dots n]$.
Ensure: A ciphertext in $\mathsf{RLWE}_{\mathfrak{z}}\big(v \cdot x^{-\mu - e}\big)$
1: $\mathtt{Acc} \leftarrow \big(0, v \cdot x^{-b}\big) \in \mathcal{R}_q^2$
2: **for** $1 \le i \le n$ **do**
3: $\quad \mathtt{Acc} \leftarrow \mathtt{Acc} + \big((x^{a_i} - 1) \cdot \mathtt{Acc}\big) \circledast \mathsf{bsk}[i]$
4: **end for**
5: **return** Acc

In order to get a better understanding of our improvements, we have to dive further into implementation details. Polynomial multiplications in the ring $(\mathbb{Z}/q\mathbb{Z})[x]/\langle x^n + 1\rangle$ are carried out with the number-theoretic transform (NTT); see e.g., [16, Chapter 8].

The external product $\circledast$ can be decomposed in two steps:

1. a gadget decomposition $\nabla_{\mathbf{g}}$, applied to both polynomial parts of Acc, and corresponding to the given bootstrapping keys, returning a vector of ℓ degree-N (small) polynomials;
2. for each of the two resulting vectors of polynomials, an inner product with the parts of the appropriate leveled component of the bootstrapping key.

For all currently known gadget decompositions, the former must be performed in the *coefficient* domain, whereas the multiplication of degree-N polynomials, where N is relatively big, requires working in the *Fourier* or *NTT* domain. Hence, the vast majority of the computational cost of the *Blind Rotation* is actually devoted to perform several forward and backward NTTs modulo the ciphertext modulus q, *at each loop iteration*.

The detailed course of operations is given in Algorithm 2. It uses an accumulator Acc and an auxiliary register Aux in the coefficient domain, both representing RLWE ciphertexts and whose respective parts are indexed by 1 and 2 respectively. Variables that live in the NTT domain are highlighted by hats, e.g., $\widehat{\mathtt{Aux}}_1 = \mathrm{NTT}_q\big(\mathtt{Aux}_1\big)$; this notation is justified by the fact that these transforms can always be done in-place. In particular, bootstrapping keys are given directly in the NTT domain as $\widehat{\mathsf{bsk}}[i]_{a,j} = \mathrm{NTT}_q\big(\mathsf{bsk}[i]_{a,j}\big)$. The Hadamard product of two values in the NTT domain, aka point-wise multiplication, is written using $\star$. Finally, the operator $\mathrm{Rot}_{\ominus}^{k}$ denotes a (right) negacyclic rotation by k positions, i.e., for any $m \in \mathcal{R}_q$ and any integer k, we have $\mathrm{Rot}_{\ominus}^{k}\, m = x^k \cdot m \pmod{x^N + 1}$.

Complexity and noise analysis: From the detailed GINX Blind Rotation in Algorithm 2, it is relatively easy to derive its computational complexity. Let $\mathsf{M}(q)$ be

Algorithm 2 GINX Blind Rotation with binary keys (detailed)

Require: Test polynomial v encoding f, bootstrapping keys $\widehat{\mathsf{bsk}}[1 \dots n]$ in the NTT domain modulo q, $\mathsf{LWE}_{\boldsymbol{s}}(\mu) = \big(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \mu + e\big)$,
Ensure: A ciphertext in $\mathsf{RLWE}_{\mathfrak{s}}\big(v \cdot x^{-\mu-e}\big)$

$\mathtt{Acc}_1, \mathtt{Acc}_2 \leftarrow \big(0, \mathrm{Rot}_{\ominus}^{-b} v\big) \in \mathcal{R}_q^2$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{s}}\big(v \cdot x^{-b}\big)$
for $1 \le i \le n$ **do**
  $\mathtt{Aux}_1, \mathtt{Aux}_2 \leftarrow \big(\mathrm{Rot}_{\ominus}^{a_i} \mathtt{Acc}_1 - \mathtt{Acc}_1,\ \mathrm{Rot}_{\ominus}^{a_i} \mathtt{Acc}_2 - \mathtt{Acc}_2\big)$ ▷ $\mathtt{Aux} = \big(x^{a_i} - 1\big) \cdot \mathtt{Acc}$
  /* Gadget Decompositions */
  $\nabla\mathtt{Aux}_1[1 \dots \ell] \leftarrow \nabla_{\mathbf{g}}\ \mathtt{Aux}_1$
  $\nabla\mathtt{Aux}_2[1 \dots \ell] \leftarrow \nabla_{\mathbf{g}}\ \mathtt{Aux}_2$ ▷ $\nabla\mathtt{Aux} = \nabla_{\mathbf{g}}\mathtt{Aux}$
  /* Inner products of polynomial vectors */
  $\widehat{\nabla\mathtt{Aux}_1}[j] \leftarrow \mathrm{NTT}_q\big(\nabla\mathtt{Aux}_1[j]\big)$ for $j = 1, \dots, \ell$
  $\widehat{\nabla\mathtt{Aux}_2}[j] \leftarrow \mathrm{NTT}_q\big(\nabla\mathtt{Aux}_2[j]\big)$ for $j = 1, \dots, \ell$
  $\widehat{\mathtt{Aux}_1}, \widehat{\mathtt{Aux}_2} \leftarrow \sum_{j=1}^{\ell} \widehat{\nabla\mathtt{Aux}_1}[j] \star \widehat{\mathsf{bsk}}[i]_{1,j} + \widehat{\nabla\mathtt{Aux}_2}[j] \star \widehat{\mathsf{bsk}}[i]_{2,j}$
    ▷ $\widehat{\mathtt{Aux}} = \mathrm{NTT}_q\big(\mathtt{Aux} \circledast \mathsf{RGSW}_{\mathfrak{s}}(s_i)\big)$
  $\mathtt{Aux}_1, \mathtt{Aux}_2 \leftarrow \big(\mathrm{iNTT}_q(\widehat{\mathtt{Aux}_1}),\ \mathrm{iNTT}_q(\widehat{\mathtt{Aux}_2})\big)$
  /* Update accumulator */
  $\mathtt{Acc}_1, \mathtt{Acc}_2 \leftarrow \big(\mathtt{Acc}_1 + \mathtt{Aux}_1,\ \mathtt{Acc}_2 + \mathtt{Aux}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{s}}\big(v \cdot x^{-b+\sum_{1 \le t \le i} a_t s_t}\big)$
end for
return $\mathtt{Acc} = \big(\mathtt{Acc}_1, \mathtt{Acc}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{s}}\big(v \cdot x^{-\mu-e}\big)$

the complexity of one modular multiplication in $\mathbb{Z}/q\mathbb{Z}$ on a w-bit word machine. For each of the n iteration of the loop, Algorithm 2 computes:

- two negacyclic rotations in $\mathcal{R}_q$, i.e., at most $4N$ additions/subtractions modulo q;
- $2N$ gadget decompositions of level ℓ of integers modulo q;
- 2ℓ forward NTTs and 2 backward iNTTs modulo q, each costing $O\big(N \log^{1+\epsilon} N \cdot \mathsf{M}(q)\big)$;
- $4\ell N \cdot \mathsf{M}(q)$ for the point-wise multiplications, using that the bootstrapping keys are given directly in the NTT domain, and $2(\ell-1)N$ additions modulo q.

Therefore, the most expensive operations are the NTT/iNTT transforms. Although, the 2ℓ NTTs (resp. the 2 iNTTs) can be done independently in parallel, thus the critical path of the whole algorithm is $n \cdot O\big(2N \log^{1+\epsilon} N \cdot \mathsf{M}(q)\big)$.

As for the noise, we refer to the thorough analysis in [11, Theorem 4.3]. For our purposes, it is sufficient to retain that for given fresh RGSW ciphertext parameters (dimension and noise distribution) and a given level of gadget decomposition, the noise distribution of the output mainly depends on the *quality* (β) of the considered gadget decomposition.

4.2 Using the Approximate CRT-Based Gadget Decomposition

In Algorithm 2, the gadget decomposition computations, when instantiated with the classical (mixed-)radix gadget decompositions, require the complete reconstruction of $\texttt{Acc}$ modulo q beforehand, which can be undesirable when q is several machine words long. On the other hand, using an (exact) CRT-based gadget decomposition requires elevating the level of the gadget decomposition, which implies an increased computational cost and bootstrapping keys size. We now show that thanks to our approximate CRT-based gadget decompositions, the whole *Blind Rotation* can be performed using only arithmetic modulo small moduli, effectively replacing *all* multi-words modular multiplications by several parallelizable smaller ones. Those units can work independently in parallel, with the only requirement that they synchronize data before and after the gadget decomposition step. We also present a modified CRT encoding of the bootstrapping keys that simplify the computation of the decomposition itself.

The resulting complete *Blind Rotation* algorithm is detailed in Algorithm 3 and thoroughly explained in the following paragraphs.

Let $q, Q = \prod_{j=1}^{\ell} q_j$, $Q_{\mathsf{low}} = \prod_{j=1}^{k} q'_j$ be as in Section 3. We further assume that we have $(\ell+k)$ arithmetic units, each of them performing arithmetic modulo its dedicated modulus. Arithmetic units handling divisors $q'_u \mid Q_{\mathsf{low}}$ (resp. $q_j \mid Q$) of the low part (resp. high) of q are called *low units* (resp. *high units*). Notation $(\|_{d\mid q}\textbf{:})$ means that the instruction can be performed independently in parallel by all arithmetic units corresponding to the subscript; conversely (**Sync:**) marks a synchronization point where units send and receive data.

The decomposition algorithm is also fixed to $\nabla \coloneqq \nabla_{\mathbf{w}}$, as defined by Equation (2), and bootstrapping keys $\mathsf{bsk}[1 \dots n]$ are now the RGSW encryptions associated to $\mathbf{w}$ of the bits of $\boldsymbol{s}$ under a key $\mathfrak{z} \in \mathcal{R}_q$.

Test polynomial and bootstrapping keys encodings: As done in Algorithm 2, the bootstrapping keys can be given directly in the NTT domain modulo q. However, we can further consider their modular reduction modulo each divisor of q, which commutes with the NTT/iNTT transform, i.e., for any $\ell \in \mathcal{R}_q$, $d \in \{q_1, \dots, q_\ell, q'_1, \dots, q'_k\}$,

$$\mathrm{NTT}_d\big(\ell \bmod d\big) = \mathrm{NTT}_q\big(\ell\big) \mod d \ .$$

Hence, each of the arithmetic units only receives a fraction of the bootstrapping keys, namely the part modulo its dedicated working modulus d, i.e., $\widehat{\mathsf{bsk}}[1 \dots n] \pmod d$.

Remark 3. Since $\sum_{1\le j\le \ell} \log q_j + \sum_{1\le u\le k} \log q'_u = \log q$, the total size of these modular keys is equivalent to the size of the original keys, especially when the moduli dividing q are specifically chosen so that their size fits one (or several) machine words.

A second transformation comes from a technique used in order to simplify the computation of our new gadget decomposition, given in Equation (2). Indeed, we

Algorithm 3 GINX Blind Rotation using approximate CRT-based gadget decomposition

Require: $\mathsf{LWE}_{\boldsymbol{s}}(\mu) = \big(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s}\rangle + \mu + e\big)$, and $\forall d \in \{q'_u\}_{u\in[\![1,k]\!]} \cup \{q_j\}_{j\in[\![1,\ell]\!]}$:
- Test polynomial $\upsilon^{(d)}$ using the modified CRT encoding as in Equation (4),
- Bootstrapping keys $\widehat{\mathsf{bsk}}[1\dots n]^{(d)}$ in the NTT domain modulo d using the modified CRT encoding as in Equation (3).

Ensure: A CRT-encoded ciphertext $\mathscr{C} \in \mathsf{RLWE}_{\mathfrak{s}}\big(\upsilon \cdot x^{-\mu-e}\big)$

/* Initialize accumulator in the modified CRT encoding (wCRT) */

$\|_{d|q}$: $\mathtt{Acc}_1^{(d)}, \mathtt{Acc}_2^{(d)} \leftarrow \big(0, \mathrm{Rot}_{\ominus}^{-b}\, \upsilon^{(d)}\big) \in \mathcal{R}_d$ ▷ $\mathtt{Acc} \in \text{wCRT}\big(\mathsf{RLWE}_{\mathfrak{s}}\big(\upsilon \cdot x^{-b}\big)\big)$

for $1 \le i \le n$ **do**

$\|_{d|q}$: $\mathtt{Aux}_1^{(d)}, \mathtt{Aux}_2^{(d)} \leftarrow \big(\mathrm{Rot}_{\ominus}^{a_i}\, \mathtt{Acc}_1^{(d)} - \mathtt{Acc}_1^{(d)},\ \mathrm{Rot}_{\ominus}^{a_i}\, \mathtt{Acc}_2^{(d)} - \mathtt{Acc}_2^{(d)}\big)$ ▷ $\mathtt{Aux} = \big(x^{a_i} - 1\big) \cdot \mathtt{Acc}$ (in wCRT)

/* Synchronized Gadget Decompositions */

Sync: Low units send $\mathtt{Aux}_1^{(q'_u)}, \mathtt{Aux}_2^{(q'_u)}$, $u \in [\![1,k]\!]$ to every high units

$\|_{q_j|Q}$: $\nabla\mathtt{Aux}_1[j] \leftarrow \mathtt{Aux}_1^{(q_j)} - \sum_{1\le u\le k} \frac{Q_{\mathsf{low}}}{q'_u} \cdot \big(\mathtt{Aux}_1^{(q'_u)}\big)_{\mathbb{Z}} \mod q_j$

$\|_{q_j|Q}$: $\nabla\mathtt{Aux}_2[j] \leftarrow \mathtt{Aux}_2^{(q_j)} - \sum_{1\le u\le k} \frac{Q_{\mathsf{low}}}{q'_u} \cdot \big(\mathtt{Aux}_2^{(q'_u)}\big)_{\mathbb{Z}} \mod q_j$ ▷ $\nabla\mathtt{Aux} = \nabla_{\mathbf{w}}\mathtt{Aux}$

Sync: Broadcast $\nabla\mathtt{Aux}[1\dots\ell]$ to obtain $\big(\nabla\mathtt{Aux}[1\dots\ell]\big)_{\mathbb{Z}} \mod d$, for all $d \mid q$.

/* Inner products of polynomial vectors: for all d dividing q */

$\|_{d|q}$: $\widehat{\nabla\mathtt{Aux}_1}[j]^{(d)} \leftarrow \mathrm{NTT}_d\big(\nabla\mathtt{Aux}_1[j] \mod d\big)$ for $j = 1,\dots,\ell$

$\|_{d|q}$: $\widehat{\nabla\mathtt{Aux}_2}[j]^{(d)} \leftarrow \mathrm{NTT}_d\big(\nabla\mathtt{Aux}_2[j] \mod d\big)$ for $j = 1,\dots,\ell$

$\|_{d|q}$: $\widehat{\mathtt{Aux}_1}^{(d)}, \widehat{\mathtt{Aux}_2}^{(d)} \leftarrow \sum_{j=1}^{\ell} \widehat{\nabla\mathtt{Aux}_1}[j]^{(d)} \star \widehat{\mathsf{bsk}}[i]_{1,j}^{(d)} + \widehat{\nabla\mathtt{Aux}_2}[j]^{(d)} \star \widehat{\mathsf{bsk}}[i]_{2,j}^{(d)}$

$\|_{d|q}$: $\mathtt{Aux}_1^{(d)}, \mathtt{Aux}_2^{(d)} \leftarrow \big(\mathrm{iNTT}_q(\widehat{\mathtt{Aux}_1}^{(d)}),\ \mathrm{iNTT}_q(\widehat{\mathtt{Aux}_2}^{(d)})\big)$ ▷ $\mathtt{Aux} \leftarrow \text{wCRT}\big(\mathtt{Aux} \circledast \mathsf{RGSW}_{\mathfrak{s}}(s_i)\big)$

/* Update all CRT shares of the accumulator */

$\|_{d|q}$: $\mathtt{Acc}_1^{(d)}, \mathtt{Acc}_2^{(d)} \leftarrow \big(\mathtt{Acc}_1^{(d)} + \mathtt{Aux}_1^{(d)},\ \mathtt{Acc}_2^{(d)} + \mathtt{Aux}_2^{(d)}\big)$ ▷ $\mathtt{Acc} \in \text{wCRT}\big(\mathsf{RLWE}_{\mathfrak{s}}\big(\upsilon \cdot x^{-b+\sum_{1\le t\le i} a_t s_t}\big)\big)$

end for

$\|_{q'_u|Q_{\mathsf{low}}}$: $\mathtt{Acc}_1^{(q'_u)}, \mathtt{Acc}_2^{(q'_u)} \leftarrow (\tau'_u)^{-1} \cdot \big(\mathtt{Acc}_1^{(d)}, \mathtt{Acc}_2^{(d)}\big)$ ▷ from wCRT to CRT

return $\mathtt{Acc} = \big(\mathtt{Acc}_1, \mathtt{Acc}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{s}}\big(\upsilon \cdot x^{-\mu-e}\big)$

remark that the twisting factors $\tau'_u \coloneqq \left(\left(\frac{Q_{\mathsf{low}}}{q'_u}\right)^{-1} \mod q'_u\right)$ do not depend on the coefficient being gadget decomposed, nor do they depend on a specific target q_j. Further, for any constant modular integer $a \in \mathbb{Z}/q'_u\mathbb{Z}$ and any polynomial $\boldsymbol{\ell} \in \mathcal{R}_{q'_u}$, we have that

$$a \cdot \mathrm{NTT}_{q'_u}(\boldsymbol{\ell}) = \mathrm{NTT}_{q'_u}(a\boldsymbol{\ell}) \mod q'_u \ .$$

Hence, we can include these factors straight into the CRT encodings of the bootstrapping keys and test polynomial modulo $q'_u \mid Q_{\mathsf{low}}$, so that when entering the gadget decomposition itself, the multiplication by τ'_u has already been taken care of by the previous steps.

Therefore, the new bootstrapping keys for our approximate CRT-based gadget decomposition are given by, for all $i \in [\![1, n]\!]$,

$$\begin{cases} \widehat{\mathsf{bsk}}[i]^{(q_j)} = \mathrm{NTT}_{q_j}\left(\quad \mathsf{bsk}[i] \mod q_j\right) & \text{for all } q_j \text{ dividing } Q \\ \widehat{\mathsf{bsk}}[i]^{(q'_u)} = \mathrm{NTT}_{q'_u}\left(\tau'_u \cdot \mathsf{bsk}[i] \mod q'_u\right) & \text{for all } q'_u \text{ dividing } Q_{\mathsf{low}} \end{cases} . \tag{3}$$

Remark 4. Due to the fact that the gadget decompositions always happen *before* incorporating the bootstrapping keys, the initialization of `Acc` also needs to include this encoding. This can be added as an explicit initialization extra step, or by requiring the test polynomial $\mathfrak{v}$ to be given in this modified CRT encoding as done in Algorithm 3, i.e., as

$$\left(\{\tau'_u \cdot \mathfrak{v} \mod q'_u\}_{u\in[\![1,k]\!]}, \{\mathfrak{v} \mod q_j\}_{j\in[\![1,\ell]\!]}\right) . \tag{4}$$

Likewise, `Acc` comes out of the loop in this modified CRT encoding, so a correction step removing the τ'_u factors is needed before returning from Algorithm 3.

Computation of our approximate CRT-based gadget decomposition: Though all arithmetic units need to be synchronized for the computation of our approximate CRT-based gadget decomposition, low and high arithmetic units have very different roles.

Using the modified CRT encoding described above, the input to the gadget decomposition is a polynomial $\boldsymbol{\ell}$, shared across low and high arithmetic units as

$$\left(\{\tau'_u \cdot \boldsymbol{\ell} \mod q'_u\}_{u\in[\![1,k]\!]}, \{\boldsymbol{\ell} \mod q_j\}_{j\in[\![1,\ell]\!]}\right) .$$

The values of $\boldsymbol{\ell} - \sum_{u=1}^{k} \frac{Q_{\mathsf{low}}}{q'_u} \cdot \left(\tau'_u \boldsymbol{\ell} \bmod q'_u\right)_{\mathbb{Z}} \mod q_j$ must be computed for all $j \in [\![1, \ell]\!]$, as described by Equation (2). This implies the following steps:

- Low units send their polynomial $\boldsymbol{\ell}'_u = \tau'_u \cdot \boldsymbol{\ell} \pmod{q_u}$ to all high units;
- Consider $j \in [\![1, \ell]\!]$; for the k incoming polynomials $\boldsymbol{\ell}'_u$, compute $\frac{Q_{\mathsf{low}}}{q'_u} \cdot \left(\boldsymbol{\ell}'_u\right)_{\mathbb{Z}} \pmod{q_j}$ (see Remark 5), and add them to the existing register containing $\boldsymbol{\ell} \pmod{q_j}$;
- At this point, each of the high units contains one of the ℓ elements of $\nabla_{\mathbf{w}}\boldsymbol{\ell}$; it remains to broadcast these ℓ polynomials *to everyone*, i.e., both to low and other high units.

We stress that, at the end of this process, every arithmetic unit contains a share of a *plain* CRT encoding of $\nabla_{\mathbf{w}} \ell$, i.e., without any additional factors τ'_u on low moduli $q'_u \mid Q_{\mathsf{low}}$.

Remark 5. Every time an integer a_1 is sent from an arithmetic unit working modulo d_1 and received by an arithmetic unit working modulo d_2, where $d_1, d_2 \mid q$, it involves an implicit lift-and-reduce operation to obtain $a_2 = \left(a_1\right)_{\mathbb{Z}} \pmod{d_2}$. Assuming all chosen moduli are equally-sized, this can be done efficiently by adding $\pm d_2$ whenever $|a_1| \geq \left\lfloor \frac{d_2}{2} \right\rfloor$, $\left\lfloor \frac{d_2}{d_1} \right\rfloor$ times at most. Ideally, all such quotients should be kept below 2, and as close to 1 as possible.

Complexity and noise analysis: Roughly speaking, using the approximate CRT-based gadget decomposition allows trading operations in $\mathbb{Z}/q\mathbb{Z}$ for operations in $\mathbb{Z}/d\mathbb{Z}$ for all of the $(k+\ell)$ chosen divisors of q. Assuming all moduli $d \in \{q'_u\} \cup \{q_j\}$ have balanced size around $\frac{\log q}{k+\ell}$, we therefore expect a gain in total bit complexity of magnitude at least

$$\frac{\mathsf{M}(q)}{\sum_{d \in \{q'_u\} \cup \{q_j\}} \mathsf{M}(d)} \approx (k+\ell)^{\omega-1} \,, \tag{5}$$

where $\mathsf{M}(d) = \lceil \log_{2^{\mathsf{w}}} d \rceil^{\omega}$ is the complexity[1] of a modular multiplication in $\mathbb{Z}/d\mathbb{Z}$ on a w-bit word machine. Likewise, the critical path is expected to shrink in similar proportions.

It remains to estimate the complexity of computing the approximate CRT-based gadget decomposition. There are 2 polynomials of degree N to gadget-decompose; to this end:

- each high unit, e.g., the one working modulo q_j, performs $k \cdot (2N)$ (negligible) lift-and-reduce operations $q'_u \to q_j$, $u \in [\![1, k]\!]$;
- each incoming polynomial $\ell'_u \pmod{q_j}$ is multiplied by the *same*, precomputed, constant $\frac{Q_{\mathsf{low}}}{q'_u} \pmod{q_j}$, i.e., $k \cdot (2N)$ modular multiplications in $\mathbb{Z}/q_j\mathbb{Z}$, $j \in [\![1, \ell]\!]$;
- broadcasting the resulting 2ℓ polynomials again involves (negligible) lift-and-reduce operations, $\ell \cdot 2(\ell-1)N$ (resp. $k \cdot 2\ell N$) on the high units (resp. low units) side.

Thus, the approximate CRT-based gadget decomposition is computationally negligible compared to the NTT/iNTT operations and inner point-wise multiplications.

Finally, the noise analysis in [11, Theorem 4.3] easily adapts to our gadget decomposition, of quality $\beta = \max_{1 \leq j \leq \ell} \left\lfloor \frac{q_j}{2} \right\rfloor$ and precision $\varepsilon \leq k \cdot \left\lfloor \frac{Q_{\mathsf{low}}}{2} \right\rfloor$ by Proposition 1.

[1] As the number of words for q is relatively small, say less than 10 at the very most, it is not unreasonable to instantiate this by $\omega = \log_2 3 \approx 1.58$ (neglecting modular reductions).

Example 1. As a concrete example, let $\mathcal{R}$ be the 2^{12}-th cyclotomic ring of degree $N = 2048$, and assume one wants to implement the *Blind Rotation* on an FPGA whose multipliers are 17 bits long [31]. The list of NTT-friendly primes $p \equiv 1 \pmod{2N}$, $p < 2^{17}$, is

$$\{12\,289,\ 40\,961,\ 61\,441,\ 65\,537,\ 86\,017,\ 114\,689\}\ .$$

A typical ciphertext modulus q in TFHE is approximately 64 bits and the (radix-based) gadget decomposition typically has precision above 30 bits. Hence, we can instantiate our approximate CRT-based gadget decomposition with $\ell = 2$, $q = q'_1 \cdot q'_2 \cdot q_1 \cdot q_2$ using

$$q'_1 = 114\,689, \quad q'_2 = 86\,017, \quad q_1 = 65\,537, \quad q_2 = 61\,441\ .$$

Note that $q = 39723809512452587521 \approx 2^{65.1}$ and that $\frac{q_{\max}}{q_{\min}} \approx 1.87$, ensuring efficient lift-and-reduce operations with at most one conditional subtraction.

Emulating a non-native multiplication modulo a 64-bit (i.e., 4 words) integer is likely to cost at least 9 multiplications of 17-bit operands with depth at least 2 or 3, plus modular reduction costs. Meanwhile, Algorithm 3 allows replacing each such multiplication by 4 *parallel* multiplications of 17-bit operands, with depth exactly one. Therefore, in practice it is expected to gain a factor $2.25 \approx 4^{0.58}$ in the total number of multiplications and running time, for an hardware usage approximately halved.

References

1. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 297–314. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_17
2. Belorgey, M.G., Carpov, S., Gama, N., Guasch, S., Jetchev, D.: Revisiting key decomposition techniques for FHE: Simpler, faster and more generic. In: Chung, K.M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 176–207. Springer (2024). https://doi.org/10.1007/978-981-96-0875-1_6
3. Bonnoron, G., Ducas, L., Fillinger, M.: Large FHE gates from tensored homomorphic accumulator. In: Joux, A., et al. (eds.) Progress in Cryptology – AFRICACRYPT 2018. LNCS, vol. 10831, pp. 217–251. Springer (2018). https://doi.org/10.1007/978-3-319-89339-6_13
4. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE instantiated with NTRU and LWE. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 188–215. Springer (2022). https://doi.org/10.1007/978-3-031-22966-4_7
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_50

6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory **6**(3), 13:1–13:36 (2014). https://doi.org/10.1145/2633600, earlier version in ITCS 2012
7. Chen, Y., Genise, N., Mukherjee, P.: Approximate trapdoors for lattices and smaller hash-and-sign signatures. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 3–32. Springer (2019). https://doi.org/10.1007/978-3-030-34618-8_1
8. Cheon, J., Costache, A., Cruz Moreno, R., Dai, W., Gama, N., Georgieva, M., Halevi, S., Kim, M., Kim, S., Laine, K., Polyakov, Y., Song, Y.: Introduction to homomorphic encryption and schemes. In: Lauter, K., Dai, W., Laine, K. (eds.) Protecting Privacy through Homomorphic Encryption, pp. 3–28. Springer (2021). https://doi.org/10.1007/978-3-030-77287-1_1
9. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_15
10. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer (2016). https://doi.org/10.1007/978-3-662-53887-6_1
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020). https://doi.org/10.1007/s00145-019-09319-x
12. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., et al. (eds.) Cyber Security Cryptography and Machine Learning (CSCML 2021). LNCS, vol. 12716, pp. 1–19. Springer (2021). https://doi.org/10.1007/978-3-030-78086-9_1
13. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 670–699. Springer (2021). https://doi.org/10.1007/978-3-030-92078-4_23
14. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_24
15. Gama, N., Izabachène, M., Nguyen, P.Q., Xie, X.: Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 528–558. Springer (2016). https://doi.org/10.1007/978-3-662-49896-519
16. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, 3rd edn. (2013). https://doi.org/10.1017/CBO9781139856065
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM Press (2009). https://doi.org/10.1145/1536414.1536440
18. Gentry, C.: Computing arbitrary functions of encrypted data. Communications of the ACM **53**(3), 97–105 (2010). https://doi.org/10.1145/1666420.1666444
19. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti,

R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_5
20. Halevi, S.: Homomorphic encryption. In: Lindell, Y. (ed.) Tutorials on the Foundations of Cryptography, pp. 219–276. Springer (2017). https://doi.org/10.1007/978-3-319-57048-8_5
21. Halevi, S., Halevi, T., Shoup, V., Stephens-Davidowitz, N.: Implementing BP-obfuscation using graph-induced encoding. In: Evans, D., et al. (eds.) 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 783–798. ACM Press (2017). https://doi.org/10.1145/3133956.3133976
22. Joye, M.: SoK: Fully homomorphic encryption over the [discretized] torus. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 661–692 (2022). https://doi.org/10.46586/tches.v2022.i4.661-692
23. Joye, M., Paillier, P.: Blind rotation in fully homomorphic encryption with extended keys. In: Dolev, S., et al. (eds.) Cyber Security, Cryptology, and Machine Learning (CSCML 2022). LNCS, vol. 13301, pp. 1–18. Springer (2022). https://doi.org/10.1007/978-3-031-07689-3_1
24. Joye, M., Walter, M.: Liberating TFHE: Programmable bootstrapping with general quotient polynomials. In: Brenner, M., et al. (eds.) 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2022). pp. 1–11. ACM Press (2022). https://doi.org/10.1145/3560827.3563376
25. Kim, M., Lee, D., Seo, J., Song, Y.: Accelerating HE operations from key decomposition technique. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 70–92. Springer (2023). https://doi.org/10.1007/978-3-031-38551-3_3
26. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_41
27. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like cryptosystems. In: Brenner, M., et al. (eds.) 9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2021). pp. 17–28. ACM Press (2021). https://doi.org/10.1145/3474366.3486924
28. Pei, D., Salomaa, A., Ding, C.: Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography. World Scientific Publishing Company (1996). https://doi.org/10.1142/3254
29. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM **56**(6), 34:1–34:40 (2009). https://doi.org/10.1145/1568318.1568324
30. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. In: DeMillo, R.A., et al. (eds.) Foundations of Secure Computation. pp. 165–179. Academic Press (1978), available at https://people.csail.mit.edu/rivest/pubs.html#RAD78
31. Xilinx: UltraScale architecture DSP slice. User Guide, v1.11 (Aug 2021), https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp