Approximate CRT-Based Gadget Decomposition and Application to TFHE Blind Rotation

Olivier Bernard and Marc Joye

Zama, Paris, France
olivier.bernard@zama.ai, marc@zama.ai

Abstract. One of the main issues to deal with for fully homomorphic encryption is the noise growth when operating on ciphertexts. To some extent, this can be controlled thanks to a so-called gadget decomposition. A gadget decomposition typically relies on radix- or CRT-based representations to split elements as vectors of smaller chunks whose inner products with the corresponding gadget vector rebuilds (an approximation of) the original elements. Radix-based gadget decompositions present the advantage of also supporting the approximate setting: for most homomorphic operations, this has a minor impact on the noise propagation but leads to substantial savings in bandwidth, memory requirements and computational costs. A typical use-case is the blind rotation as used for example in the bootstrapping of the TFHE scheme. On the other hand, CRT-based representations are convenient when machine words are too small for directly accommodating the arithmetic on large operands. This arises in two typical cases: (i) in the hardware case with multipliers of restricted size, e.g., 17 bits; (ii) in the software case for ciphertext moduli above, e.g., 128 bits.
This paper presents new CRT-based gadget decompositions *for the approximate setting*, which combines the advantages of non-exact decompositions with those of CRT-based decompositions. Significantly, it enables certain hardware or software realizations otherwise hardly supported like the two aforementioned cases. In particular, we show that our new gadget decompositions provide implementations of the (programmable) bootstrapping in TFHE relying *solely* on native arithmetic and offering extra degrees of parallelism.

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

1 Introduction

Fully homomorphic encryption (in short, FHE) [RAD78, Gen10] is often referred to as the 'holy grail of cryptography.' Contrary to traditional encryption technologies, FHE encryption allows anyone to directly perform operations on encrypted data, without the need of decrypting them beforehand from their processing. The result of the computation is encrypted. We refer the reader to [Hal17, CCC+21] for surveys on fully homomorphic encryption.

In turn, FHE gives rise to the paradigm of end-to-end encryption. Let Enc be an homomorphic encryption scheme. In the common setting of a client and a server, the client encrypts some private data $x_1, \ldots, x_u$ under its key sk and sends the corresponding ciphertexts $C_1 \leftarrow \mathrm{Enc}_{sk}(x_1), \ldots, C_u \leftarrow \mathrm{Enc}_{sk}(x_u)$ to the server. The server homomorphically processes $C_1, \ldots, C_u$ on a circuit representing some functionality f and gets as a result $\hat{C} \leftarrow \mathrm{Enc}_{sk}(f(x_1, \ldots, x_u))$. Ciphertext $\hat{C}$ is returned to the client which can then decrypt it using sk to get $f(x_1, \ldots, x_u)$. It is worth observing that the server learnt nothing on

private inputs $x_1, \ldots, x_u$, nor on the output result $f(x_1 \ldots, x_u)$. In the medium-to-long term, one could imagine Internet traffic being FHE-encrypted through an `httpz` protocol, thereby preserving the privacy of users accessing websites, when their data is in transit as currently offered by `https` (SSL/TLS)—but also when it is processed! Think for example of your favorite search engine operating on encrypted keywords and obtaining encrypted URLs of relevant web-pages.

Apart from a few exceptions, most known instantiations for FHE rely on lattices, basing their security on the learning with errors (LWE) problem [Reg09] or variants thereof. As a consequence, for security reasons, the corresponding ciphertexts must be noisy. While this is in general not an issue for regular encryption, this must be dealt with care in the case of fully homomorphic encryption. The problem is that the noise present in the ciphertexts tends to grow when noisy ciphertexts are homomorphically processed. If the noise grows above a certain threshold, ciphertexts can no longer be decrypted. There are basically two ways to address this problem: *(i)* bootstrapping ciphertexts and *(ii)* controlling the noise growth in ciphertexts. The approach of *bootstrapping* was introduced in Gentry's seminal work in 2009 [Gen09]. It consists in homomorphically evaluating the decryption circuit on input an encryption of a ciphertext and of the decryption key, yielding another ciphertext that encrypts the same plaintext—this 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. A complementary approach 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 the need of bootstrapping. A well-known trick is the *gadget decomposition* [MP12, BGV14]: for multiplying a noisy ciphertext by a scalar, the scalar is first decomposed with respect to a small radix B. Specifically, if Enc denotes an 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., [BGV14, MP12, Bra12, GSW13, AP14, DM15, GINX16, CKKS17, CGGI20, BIP+22]. Of special importance is the gadget decomposition when applied to improve bootstrapping procedures. In particular, similarly to [AP14, DM15], the bootstrapping in the TFHE scheme, building on [GINX16], 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 [CGGI20]. 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 [CJP21]. 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 function. A detailed description of the programmable bootstrapping with companion algorithms can be found in [Joy22].

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 significantly 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 [HHSS17, Section 4.2].

Alternative gadget decompositions have been considered, including representations relying on the Chinese Remainder Theorem (CRT) [BDF18, Section B.4]. Operations modulo the small factors can also be grouped in a two-level way, as demonstrated in [KLSS23]. This is mostly useful for large ciphertext moduli as in CKKS-like schemes [CKKS17]; see also [BCG+23] for an extension using a bivariate polynomials formalism.

Chinese remaindering is a natural method for handling large integers using small arithmetic chunks but it oughts 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 contributions 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.

Applications of our approximate CRT-based gadget decompositions to the use-cases of machine learning and of threshold FHE decryption are also examined.

Outline of the paper The rest of this paper is organized as follows. The next section abstracts the notion of gadget decomposition and the different flavors of gadget decomposition. It also introduces relevant building blocks towards applications to TFHE and the likes. Section 3 is the core of the paper. It proposes and specifies approximate CRT-based gadget decompositions. The gadget decompositions are applied to the blind rotation in Section 4, at the heart of the programmable bootstrapping procedure. Other applications are also discussed. Finally, Section 5 concludes the paper.

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. For example, for $q = 5$, $\lfloor \frac{5}{2} \rfloor = 2$ and elements of $\mathbb{Z}/5\mathbb{Z}$ are represented by the set $\{-2, -1, 0, 1, 2\}$. 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 $(a \bmod q)_{\mathbb{Z}}$ or sometimes, more simply, as $(a)_{\mathbb{Z}}$. 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 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, 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 $\mathcal{R}_q = \mathbb{Z}/q\mathbb{Z}$;
- $\mathcal{K} = \mathbb{Q}(\zeta_m) \cong \mathbb{Q}[x]/\langle \Phi_m(x) \rangle$, the m-th cyclotomic field, 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 [CGGI20, 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 $\mathscr{a} \in \mathcal{R}_q$:

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

   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 $\mathbb{T}_q := \frac{1}{q}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T} := \mathbb{R}/\mathbb{Z}$ 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. [Joy22, Remark 3]. Alternatively, the gadget algorithm with parameters $(\ell, \beta, \varepsilon)$ can be directly defined as $\nabla_{\mathbf{g}}^{\beta,\varepsilon} \colon \mathbb{T}_q[x]/\langle \Phi_m(x) \rangle \to \left(\mathbb{Z}[x]/\langle \Phi_m(x) \rangle\right)^\ell$ for some gadget vector $\mathbf{g} \in \left(\mathbb{T}_q[x]/\langle \Phi_m(x) \rangle\right)^\ell$, viewing $\mathbb{T}[x]/\langle \Phi_m(x) \rangle$ as a $\mathbb{Z}[x]/\langle \Phi_m(x) \rangle$-module.

2.1.1 Radix-based gadget decomposition

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

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 $\beta = \frac{B}{2}$; specifically, we write $a \equiv \sum_{j=1}^{\ell} a_j \frac{q}{B^j} + R \pmod q$ with $-B/2 \leq a_j \leq B/2$ 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 $a - \langle \nabla a, \mathbf{g} \rangle \equiv a - \sum_{j=1}^{\ell} a_j \frac{q}{B^j} \equiv R \pmod q$ and $|R| \leq \lfloor q/(2B^\ell) \rfloor$. It is worth remarking that $\varepsilon = 0$ when $q = B^\ell$.

Example 1. Take $q = 2^{32}$, $B = 64$, and $\ell = 4$. Suppose $a = 3141592653$ and $\mathbf{g} = (2^{26}, 2^{20}, 2^{14}, 2^8)$. Then $\nabla a = (-17, -12, 4, -26)$ and $|R| = 17 \leq \varepsilon = 2^7$.

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 $\mathbf{g} = \left(\frac{q}{q_1}, \frac{q}{q_1 q_2} \ldots, \frac{q}{q_1 q_2 \cdots q_\ell}\right)$. The quality is of $\beta = \lfloor \max_j q_j/2 \rfloor$ and the precision is of $\varepsilon = \lfloor \frac{q}{2Q} \rfloor$. Radix-$B$ gadget decompositions correspond to the special case $q_1 = q_2 = \cdots = q_\ell = B$.

2.1.2 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

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

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

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 $\beta = \max_i \lfloor \frac{q_i}{2} \rfloor$.

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

Example 2. Take $q_1 = 2^8 - 1$, $q_2 = 2^8$, $q_3 = 2^8 + 1$, $q_4 = 2^8 + 3$, and $q = q_1 q_2 q_3 q_4 = 4345232640$. This gives rise to $\mathbf{z} = (545284096, 1442753025, -1082081280, -905955840)$. For $a = 3141592653$, one gets $\nabla_{\mathbf{z}} a = (48, 77, -19, 94)$ and $a - \langle \nabla_{\mathbf{z}} a, \mathbf{z} \rangle \equiv 0 \pmod q$.

Suppose now a faulty gadget decomposition $\nabla'_{\mathbf{z}} a = (48, 77, -19, 93)$ on the last component. Then $a - \langle \nabla'_{\mathbf{z}} a, \mathbf{z} \rangle \equiv z_4 \equiv -905955840 \pmod q$.

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

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

are formed. The vector $\nabla_{\mathbf{z}}\mathscr{f} = \big(\mathscr{f}_1, \ldots, \mathscr{f}_\ell\big) \in \mathcal{R}^\ell$ represents the CRT-based gadget decomposition of $\mathscr{f}$. 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 $\mathscr{f} - \big\langle \nabla_{\mathbf{z}}\mathscr{f}, \mathbf{z} \big\rangle \equiv 0 \pmod q$, and thus $\varepsilon = 0$. Further, if $\beta = \max_i \big\lfloor \frac{q_i}{2} \big\rfloor$ then $\big\| \nabla_{\mathbf{z}}\mathscr{f} \big\|_\infty \leq \beta$, where the infinity norm of a polynomial is defined as the infinity norm of the vector of its coefficients.

2.2 Fully Homomorphic Encryption

2.2.1 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 $\mathfrak{z} \in \mathcal{R}^k$, a *generalized LWE sample* is a vector of the form

$$\big(\mathscr{a} = (\mathscr{a}_1, \ldots, \mathscr{a}_k), \mathscr{r}\big) \in \mathcal{R}_q^{k+1} \quad \text{where } \mathscr{r} = \langle \mathscr{a}, \mathfrak{z} \rangle + \mathscr{e}$$

with $\mathscr{a} \xleftarrow{\$} \mathcal{R}_q^k$ and $\mathscr{e} \leftarrow \chi$. The generalized LWE assumption posits that such a sample is indistinguishable from a uniformly random vector in $\mathcal{R}_q^{k+1}$. This complexity assumption can serve as a basis to build semantically secure ciphertexts. Vector $\mathfrak{z}$ plays the role of the encryption key and $\mathscr{r}$ is used as an additive one-time to conceal the message to encrypt. Specifically, given a fresh sample $(\mathscr{a}, \mathscr{r}) \in \mathcal{R}_q^{k+1}$, (the encoding of) a message μ in $\mathcal{R}_q$, called plaintext, is encrypted under key $\mathfrak{z}$ to form the ciphertext

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

Two specialized instances are typically used:

1. $\mathcal{R}_q \cong \mathbb{Z}[x]/\langle x^N + 1 \rangle$ 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.

Remark 1. In the RLWE setting, we write $\mathscr{C} \leftarrow \mathsf{RLWE}_{\mathfrak{z}}(\mu) \in \big(\mathbb{Z}[x]/\langle x^N + 1 \rangle\big)^2$ to denote the encryption of $\mu \in \mathbb{Z}[x]/\langle x^N + 1 \rangle$ under key $\mathfrak{z}$. Likewise, in the LWE setting, letting $\boldsymbol{s} = \mathfrak{z}$ and $n = k$, we write $\boldsymbol{C} \leftarrow \mathsf{LWE}_{\boldsymbol{s}}(\mu) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ for the encryption of $\mu \in \mathbb{Z}/q\mathbb{Z}$ under key $\boldsymbol{s}$.

2.2.2 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

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

and its GGSW expansion

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

Following [MP21], 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:

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

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

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

$$\begin{aligned}
\alpha \odot \mathsf{GLev}_{\mathfrak{s}}^{\mathbf{g}}(m) &:= \big\langle \nabla\alpha, \mathsf{GLev}_{\mathfrak{s}}^{\mathbf{g}}(m) \big\rangle \\
&= \textstyle\sum_{j=1}^{\ell} \alpha_j \cdot \mathsf{GLWE}_{\mathfrak{s}}(g_j \cdot m) = \mathsf{GLWE}_{\mathfrak{s}}\Big(\big(\sum_{j=1}^{\ell} \alpha_j \cdot g_j\big) \cdot m\Big) \\
&= \mathsf{GLWE}_{\mathfrak{s}}\big(\big\langle \nabla_{\mathbf{g}}\alpha, \mathbf{g} \big\rangle \cdot m\big) = \mathsf{GLWE}_{\mathfrak{s}}\big((\alpha \odot \mathbf{g}) \cdot m\big) \\
&= \mathsf{GLWE}_{\mathfrak{s}}(\alpha \cdot m) \ .
\end{aligned}$$

One so gets $\mathsf{GLev}_{\mathfrak{s}}^{\mathbf{g}}(\alpha \cdot m)$ as an output by evaluating $\big((\alpha \cdot g_j) \odot \mathsf{GLev}_{\mathfrak{s}}^{\mathbf{g}}(m)\big)_{1 \le j \le \ell}$.

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

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

where $\big(a_1, \ldots, a_k, b = \sum_{j=1}^{k} a_j \cdot \mathfrak{s}_j + \mu_1 + e\big)$ expands the input $\mathsf{GLWE}_{\mathfrak{s}}(\mu_1)$. The result is a GLWE encryption of $\mu_1 \cdot m_2$ if message m_2 is small so that $\|e \cdot m_2\|_\infty \approx \|e\|_\infty$. 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 [PSD96, 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 2.1.2, any $\ell \in \mathcal{R}_q$ may be expressed *exactly* as $\ell = \langle \nabla_{\mathbf{z}} \ell, \mathbf{z} \rangle \mod q$. However, some applications only require an approximate expression $\tilde{\ell}$ for ℓ, provided that $\tilde{\ell}$ satisfies $\left\| \ell - \tilde{\ell} \right\|_\infty \leq \varepsilon$ for some given bound ε.

A typical 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 $\mu = \lfloor q/t \rceil m$ 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

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

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 a polynomial $f = \sum_{i=0}^{N-1} f_i\, x^i \in \mathcal{R}_q$ is then given by the ℓ-tuple

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

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

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

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 $\left\{ \frac{Q_{\mathsf{low}}}{q'_1} \mod q_j, \ldots, \frac{Q_{\mathsf{low}}}{q'_k} \mod q_j \right\}$ 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* $f \in \mathcal{R}_q$*:*

$$\left\|\nabla_{\mathbf{w}} f\right\|_\infty \leq \beta = \max_{1 \leq j \leq \ell} \left\lfloor \tfrac{q_j}{2} \right\rfloor$$

and

$$\left\| f - \left\langle \nabla_{\mathbf{w}} f, \mathbf{w} \right\rangle \right\|_\infty \leq \varepsilon = k \cdot \left\lfloor \tfrac{Q_{\mathsf{low}}}{2} \right\rfloor ,$$

where the infinity norms are understood coefficient-wise.

Remark 2. 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. The only non-immediate statement is relative to the precision of the gadget decomposition. Let $\tilde{f} \coloneqq \left\langle \nabla_{\mathbf{w}} f, \mathbf{w} \right\rangle = \sum_{j=1}^{\ell} w_j \cdot f_j \mod q$. Extracting the Q_{low} factor from the w_j's yields that $\tilde{f}$ can be written as $Q_{\mathsf{low}} \cdot \left(\mathcal{F} \bmod Q\right)_{\mathbb{Z}}$, where

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

Thus, by the CRT applied to the high part using $\gcd\left(Q_{\mathsf{low}}, Q\right) = 1$, for all $j \in [\![1, \ell]\!]$ we have that $\mathcal{F} \equiv \frac{f_j}{Q_{\mathsf{low}}} \pmod{q_j}$. Now, let $\mathcal{S}$ be the polynomial whose coefficients are given by the inner sum indexed by u in Equation (2), i.e.,

$$\mathcal{S} \coloneqq \sum_{u=1}^{k} \widetilde{Q}'_u \cdot \left(\left(\widetilde{Q}'_u\right)^{-1} \cdot f \mod q'_u \right)_{\mathbb{Z}} ,$$

where $\widetilde{Q}'_u = \frac{Q_{\mathsf{low}}}{q'_u}$ for $u \in [\![1, k]\!]$, so that $f_j \equiv f - \mathcal{S} \pmod{q_j}$ for all $j \in [\![1, \ell]\!]$. The first key observation about $\mathcal{S}$ is that, by the CRT applied to divisors of Q_{low}, $\mathcal{S} \equiv f \pmod{Q_{\mathsf{low}}}$. Hence, $\left(f - \mathcal{S}\right)$ is actually divisible by Q_{low}, which in turn implies

$$\begin{aligned} \tilde{f} &= Q_{\mathsf{low}} \cdot \left(\mathcal{F} \mod Q\right)_{\mathbb{Z}} \\ &= Q_{\mathsf{low}} \cdot \left(\tfrac{f - \mathcal{S}}{Q_{\mathsf{low}}} \mod Q \right)_{\mathbb{Z}} = f - \mathcal{S} \mod q \ . \end{aligned}$$

The second key observation about $\mathcal{S}$ is that its coefficients have amplitude bounded by $\|\mathcal{S}\|_\infty = \left\|f - \tilde{f}\right\|_\infty \leq k \cdot \left\lfloor \frac{Q_{\mathsf{low}}}{2} \right\rfloor$, yielding the result. □

Remark 3. 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.

Example 3. Suppose $\beta = 256$ and modulus q is a 32-bit integer, which is divided in $q'_1 = 233$, $q'_2 = 239$ for the low part ($Q_{\mathsf{low}} = 55687$) and $q_1 = 241$, $q_2 = 251$ for the high part ($Q = 60491$); hence $q = 3368562317$ and $\ell = 2$. The corresponding gadget vector is

$$\mathbf{w} = \big(1663315003, 952860257\big) \in \mathcal{R}_q^2 \ .$$

In Equation (2), we have that $\Big(\big(\frac{Q_{\mathsf{low}}}{q'_u}\big)^{-1} \bmod q'_u\Big)_{\mathbb{Z}}$ is 39 for $u = 1$ (resp. -40 for $u = 2$). If the input polynomial ℓ is

$$\ell = 1618033988\, x^3 + 749894848\, x^2 - 1322693974\, x + 656381177 \ ,$$

then $\nabla_{\mathbf{w}} \ell = \big(\ell_1, \ell_2\big) \in \mathcal{R}^2$ with

$$\begin{cases} \ell_1 = \big(\ell \bmod 241\big) + 2 \cdot \big(39\ell \bmod 233\big)_{\mathbb{Z}} + 8 \cdot \big(-40\ell \bmod 239\big)_{\mathbb{Z}} \mod 241 \\ \quad = 7\, x^3 + 2\, x^2 + 9\, x - 111, \\ \ell_2 = \big(\ell \bmod 251\big) + 12 \cdot \big(39\ell \bmod 233\big)_{\mathbb{Z}} + 18 \cdot \big(-40\ell \bmod 239\big)_{\mathbb{Z}} \mod 251 \\ \quad = 92\, x^3 + 68\, x^2 + 43\, x + 99 \ . \end{cases}$$

The corresponding approximate polynomial $\tilde{\ell} \coloneqq \big\langle \nabla_{\mathbf{w}} \ell, \mathbf{w} \big\rangle$ is equal to

$$\tilde{\ell} = 1618041472\, x^3 + 749881142\, x^2 - 1322733311\, x + 656382669 \ ,$$

whose distance from ℓ in infinity-norm (coefficient-wise) is

$$\begin{aligned} \big\|\ell - \tilde{\ell}\big\|_\infty &= \big\|-7484\, x^3 + 13706\, x^2 + 39337\, x - 1492\big\|_\infty \\ &= 39337 \leq 2 \cdot \big\lfloor \tfrac{55687}{2} \big\rfloor = 55686 \ . \end{aligned}$$

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, e.g., ternary, other rings $\mathcal{R}$, 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 of $m \in \mathbb{Z}/t\mathbb{Z}$, i.e., from

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

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 $\mathbf{g} = (g_1, \ldots, g_\ell) \in \mathcal{R}_q^\ell$. 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

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

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 The programmability of GINX bootstrapping comes from the so-called *test polynomial*. 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., $f(x) = -f\left(x + \frac{t}{2}\right)$. The test polynomial can be then defined as

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

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 $\mathfrak{v} \cdot x^{-\lfloor 2N/t \rceil m - e}$, 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 $\mathscr{m}$ and outputs a RLWE encryption $\mathscr{C}'$ of $x^{s_i a_i} \cdot \mathscr{m}$. Concretely, this is achieved by computing

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

noting that $x^{s_i a_i} \cdot \mathscr{m}$ is equal to $\mathscr{m}$ if $s_i = 0$, and to $x^{a_i} \cdot \mathscr{m}$ 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 $\mathtt{Acc} = \left(0, \mathfrak{v} \cdot x^{-b}\right) \in \mathcal{R}_q^2$, 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: $\mathsf{LWE}_{\boldsymbol{s}}\left(\lfloor \frac{2N}{t} \rceil m\right) = \left(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \lfloor \frac{2N}{t} \rceil m + e\right)$, bootstrapping keys $\mathsf{bsk}[1 \ldots n]$.
Ensure: A ciphertext in $\mathsf{RLWE}_{\mathfrak{z}}\left(\mathfrak{v} \cdot x^{-\lfloor 2N/t \rceil m - e}\right)$

```
Acc ← (0, v · x^{-b}) ∈ R_q^2
for 1 ≤ i ≤ n do
    Acc ← Acc + ((x^{a_i} − 1) · Acc) ⊛ bsk[i]
end for
return Acc
```

In order to get a better understanding of our improvements, we have to dive further into implementation details. Polynomial multiplications in $(\mathbb{Z}/q\mathbb{Z})[x]/\langle x^n + 1 \rangle$ are carried out with the number-theoretic transform (NTT); see e.g., [vzGG13, Chapter 8].

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

1. a gadget decomposition $\nabla_{\mathbf{g}}$, applied to both mask and body of $\mathtt{Acc}$, 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 body and mask 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 $\mathtt{Acc}$ and an auxiliary register $\mathtt{Aux}$ in the coefficient domain, both representing RLWE ciphertexts and whose respective masks and bodies 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 $\mathrm{Rot}_{\ominus}^{k}\, m = x^k \cdot m \pmod{x^N+1}$.

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, $\mathsf{LWE}_{\boldsymbol{s}}\big(\lfloor\frac{2N}{t}\rceil m\big) = \big(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s}\rangle + \lfloor\frac{2N}{t}\rceil m + e\big)$,
Ensure: A ciphertext in $\mathsf{RLWE}_{\mathfrak{z}}\big(v \cdot x^{-\lfloor 2N/t\rceil \mu - e}\big)$

$\mathtt{Acc}_1, \mathtt{Acc}_2 \leftarrow \big(0, \mathrm{Rot}_{\ominus}^{-b}\, v\big) \in \mathcal{R}_q^2$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{z}}\big(v \cdot x^{-b}\big)$
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 $j = 1, \dots, \ell$
  $\widehat{\mathtt{Aux}}_1, \widehat{\mathtt{Aux}}_2 \leftarrow \sum_{j=1}^{\ell} \widehat{\nabla\mathtt{Aux}}_1[j] \star \widehat{\mathsf{bsk}}[i]_{1,j} + \widehat{\nabla\mathtt{Aux}}_2[j] \star \widehat{\mathsf{bsk}}[i]_{2,j}$
    ▷ $\widehat{\mathtt{Aux}} = \mathrm{NTT}_q\big(\mathtt{Aux} \circledast \mathsf{RGSW}_{\mathfrak{z}}(s_i)\big)$
  $\mathtt{Aux}_1, \mathtt{Aux}_2 \leftarrow \big(\mathrm{iNTT}_q(\widehat{\mathtt{Aux}}_1),\ \mathrm{iNTT}_q(\widehat{\mathtt{Aux}}_2)\big)$
  /* Update accumulator */
  $\mathtt{Acc}_1, \mathtt{Acc}_2 \leftarrow \big(\mathtt{Acc}_1 + \mathtt{Aux}_1,\ \mathtt{Acc}_2 + \mathtt{Aux}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{z}}\big(v \cdot x^{-b + \sum_{1 \le t \le i} a_t s_t}\big)$
end for
return $\mathtt{Acc} = \big(\mathtt{Acc}_1, \mathtt{Acc}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{z}}\big(v \cdot x^{-\lfloor 2N/t\rceil \mu - e}\big)$

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 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 [CGGI20, 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 `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 ($\|_{d|q}$:) 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 $\ell \in \mathcal{R}_q$, $d \in \{q_1, \dots, q_\ell, q'_1, \dots, q'_k\}$,

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

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., $\widehat{\mathsf{bsk}}[1 \dots n] \pmod d$.

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

Require: $\mathsf{LWE}_{\boldsymbol{s}}\big(\lfloor\frac{2N}{t}\rceil m\big) = \big(\boldsymbol{a}, b = \langle\boldsymbol{a},\boldsymbol{s}\rangle + \lfloor\frac{2N}{t}\rceil m + e\big)$, and $\forall d \in \big\{q'_u\big\}_{u\in[\![1,k]\!]} \cup \big\{q_j\big\}_{j\in[\![1,\ell]\!]}$:

- Test polynomial $\mathfrak{v}^{(d)}$ 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 $\mathscr{C} \in \mathsf{RLWE}_{\mathfrak{z}}\big(\mathfrak{v}\cdot x^{-\lfloor 2N/t\rceil m - e}\big)$

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

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

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 $j = 1,\dots,\ell$

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

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

  /* Update all CRT shares of the accumulator */

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

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 $\mathtt{Acc} = \big(\mathtt{Acc}_1, \mathtt{Acc}_2\big)$ ▷ $\mathtt{Acc} \in \mathsf{RLWE}_{\mathfrak{z}}\big(\mathfrak{v}\cdot x^{-\lfloor 2N/t\rceil\mu - e}\big)$

Remark 4. 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 remark that the twisting factors $\tau'_u \coloneqq \Big(\big(\frac{Q_{\mathsf{low}}}{q'_u}\big)^{-1} \mod q'_u\Big)$ 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 $\ell \in \mathcal{R}_{q'_u}$, we have that

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

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 $i \in [\![1, n]\!]$,

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

Remark 5. 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 v to be given in this modified CRT encoding as done in Algorithm 3, i.e., as

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

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 $\mathfrak{f}$, shared across low and high arithmetic units as

$$\Big(\big\{\tau'_u \cdot \mathfrak{f} \mod q'_u\big\}_{u\in[\![1,k]\!]}, \big\{\mathfrak{f} \mod q_j\big\}_{j\in[\![1,\ell]\!]}\Big) .$$

We must compute, for all $j \in [\![1, \ell]\!]$, $\mathfrak{f} - \sum_{u=1}^{k} \frac{Q_{\mathsf{low}}}{q'_u} \cdot \big(\tau'_u \mathfrak{f} \bmod q'_u\big)_{\mathbb{Z}} \mod q_j$, as described by Equation (2). This implies the following steps:

- Low units send their polynomial $\mathfrak{f}'_u = \tau'_u \cdot \mathfrak{f} \pmod{q_u}$ to all high units;
- Consider $j \in [\![1, \ell]\!]$; for the k incoming polynomials $\mathfrak{f}'_u$, compute $\frac{Q_{\mathsf{low}}}{q'_u} \cdot \big(\mathfrak{f}'_u\big)_{\mathbb{Z}} \pmod{q_j}$ (see Remark 6), and add them to the existing register containing $\mathfrak{f} \pmod{q_j}$;
- At this point, each of the high units contains one of the ℓ elements of $\nabla_{\mathbf{w}}\mathfrak{f}$; 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 $\nabla_{\mathbf{w}}\mathfrak{f}$, i.e., without any additional factors τ'_u on low moduli $q'_u \mid Q_{\mathsf{low}}$.

Remark 6. 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 $a_2 = \big(a_1\big)_{\mathbb{Z}} \pmod{d_2}$. Assuming all chosen moduli are equally-sized, this can be done efficiently by adding $\pm d_2$ whenever $\big|a_1\big| \geq \big\lfloor \frac{d_2}{2} \big\rfloor$, $\big\lfloor \frac{d_2}{d_1} \big\rfloor$ 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 $\mathsf{M}(d) = \big\lceil \log_{2^{\mathsf{w}}} d \big\rceil^{\omega}$ 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.

[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).

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 $f'_u \pmod{q_j}$ 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 [CGGI20, Theorem 4.3] can be easily adapted to our gadget decomposition, of quality $\beta = \max_{1 \le j \le \ell} \left\lfloor \frac{q_j}{2} \right\rfloor$ and precision $\varepsilon \le k \cdot \left\lfloor \frac{Q_{\mathsf{low}}}{2} \right\rfloor$ by Proposition 1.

Example 4. 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 [Xil21]. 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 $\frac{q_{\mathsf{max}}}{q_{\mathsf{min}}} \approx 1.87$, 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.

4.3 Further Use-Cases

4.3.1 Homomorphic machine learning

A critical operation for machine learning on encrypted data consists in computing homomorphically many weighted sums of the form

$$\sum_{t=1}^{M} \omega_t \cdot \mathscr{C}_t\ ,$$

which serve as inputs to a non-linear function; see e.g., [CJP21]. Each term of this sum involves a scalar product between a (possibly small) weight ω_t and a ciphertext $\mathscr{C}_t$, involving as much gadget decompositions. Those operations are to be accelerated on dedicated, massively parallel, hardware such as GPUs, FPGAs or ASICs.

These architectures can have quite small arithmetic units. For example, FPGAs typically contain multipliers ranging from 17 bits to 23 bits, depending on the model [Xil21]; on the other hand, GPUs might take advantage of performing vectorized operations on several 16 bits or 32 bits operands for a few thousands of parallel threads.

Our approximate CRT-based gadget decompositions allow taking full advantage of the intrinsic computational characteristics of these hardwares, with a simple and easily parallelizable work-flow.

4.3.2 Threshold decryption for TFHE

In the context of threshold decryption for FHE, and in particular for blockchain applications, a standard technique to avoid leaking to the parties the shared secret key through the noise is to inject some additional noise into the secret shares, a technique which is called *noise flooding*.

For TFHE, where the ciphertext modulus is usually relatively small (say, 64 bits or less) to enhance performance, this technique is not directly applicable as the noise gap is too small to preserve the message from being destroyed during the noise flooding operation. Therefore, a technique called *Switch-n-Squash* was proposed in [DDK+23], that embeds ciphertexts into larger parameters during a bootstrapping operation; thereafter the regular noise flooding operation can be safely applied.

As a consequence, costly bootstrapping operations, such as NTTs, or more generally polynomial multiplications, now have to be computed on larger integers, e.g., 128 bits as suggested in [DDK+23, Table 1]. In practice, optimized code for these sizes is not readily available off the shelf and moving beyond the frontier of a single machine word incurs a noticeable computational overhead.

Our approximate CRT-based gadget decompositions make it possible to reuse code that is optimized for 64-bit words for the extended ciphertext moduli featured in [DDK+23, Table 1]. This can be done in parallel, with no additional overhead apart from the synchronization step when computing the gadget decomposition.

5 Conclusion

This paper introduced CRT-based gadget decompositions in the approximate setting. Complete specifications and related parameters were provided and discussed, enabling homomorphic operations otherwise hardly available in certain hardware/software configurations. In particular, dedicated implementations of the blind rotation as used in the programmable bootstrapping were reported and thoroughly analyzed. Additional benefits were also reported. Finally, further applications to homomorphic machine learning and threshold FHE decryption were reviewed.

References

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2014. doi:10.1007/978-3-662-44371-2_17.

[BCG+23] Mariya Georgieva Belorgey, Sergiu Carpov, Nicolas Gama, Sandra Guasch, and Dimitar Jetchev. Revisiting key decomposition techniques for FHE: Simpler, faster and more generic. Cryptology ePrint Archive, 2023. URL: https://ia.cr/2023/771.

[BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In A. Joux, A. Nitaj, and T. Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2018*, volume 10831 of *Lecture Notes in Computer Science*, pages 217–251. Springer, 2018. doi:10.1007/978-3-319-89339-6_13.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on*

Computation Theory, 6(3):13:1–13:36, 2014. Earlier version in ITCS 2012. doi:10.1145/2633600.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. In S. Agrawal and D. Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, volume 13792 of *Lecture Notes in Computer Science*, pages 188–215. Springer, 2022. doi:10.1007/978-3-031-22966-4_7.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012. doi:10.1007/978-3-642-32009-5_50.

[CCC+21] Jung Hee Cheon, Anamaria Costache, Radames Cruz Moreno, Wei Dai, Nicolas Gama, Mariya Georgieva, Shai Halevi, Miran Kim, Sunwoong Kim, Kim Laine, Yuriy Polyakov, and Yongsoo Song. Introduction to homomorphic encryption and schemes. In K. Lauter, W. Dai, and K. Laine, editors, *Protecting Privacy through Homomorphic Encryption*, pages 3–28. Springer, 2021. doi:10.1007/978-3-030-77287-1_1.

[CGGI20] Ilaria Chilotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. doi:10.1007/s00145-019-09319-x.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In S. Dolev et al., editors, *Cyber Security Cryptography and Machine Learning (CSCML 2021)*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021. doi:10.1007/978-3-030-78086-9_1.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

[DDK+23] Morten Dahl, Daniel Demmler, Sarah El Kazdady, Arthur Meyre, Jean-Baptiste Orfila, Dragor Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. Noah's Ark: Efficient threshold-FHE using noise flooding. In M. Brenner, A. Costache, and K. Rohloff, editors, *11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2023)*, pages 35–46. ACM, 2023. doi:10.1145/3605759.3625259.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, 2009. doi:10.1145/1536414.1536440.

[Gen10] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010. doi:10.1145/1666420.1666444.

[GINX16] Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 528–558. Springer, 2016. doi:10.1007/978-3-662-49896-519.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013. doi:10.1007/978-3-642-40041-4_5.

[Hal17] Shai Halevi. Homomorphic encryption. In Y. Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 219–276. Springer, 2017. doi:10.1007/978-3-319-57048-8_5.

[HHSS17] Shai Halevi, Tzipora Halevi, Victor Shoup, and Noah Stephens-Davidowitz. Implementing BP-obfuscation using graph-induced encoding. In D. Evans, T. Malkin, and D. Xu, editors, *2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 783–798. ACM Press, 2017. doi:10.1145/3133956.3133976.

[Joy22] Marc Joye. SoK: Fully homomorphic encryption over the [discretized] torus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):661–692, 2022. doi:10.46586/tches.v2022.i4.661-692.

[KLSS23] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Accelerating HE operations from key decomposition technique. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 70–92. Springer, 2023. doi:10.1007/978-3-031-38551-3_3.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012. doi:10.1007/978-3-642-29011-4_41.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In M. Brenner et al., editors, *9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2021)*, pages 17–28. ACM Press, 2021. doi:10.1145/3474366.3486924.

[PSD96] Dingyi Pei, Arto Salomaa, and Cunsheng Ding. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Company, 1996. doi:10.1142/3254.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In R. A. DeMillo et al., editors, *Foundations of Secure Computation*, pages 165–179. Academic Press, 1978. Available at https://people.csail.mit.edu/rivest/pubs.html#RAD78.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34:1–34:40, 2009. doi:10.1145/1568318.1568324.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013. doi:10.1017/CBO9781139856065.

[Xil21] Xilinx. UltraScale architecture DSP slice. User Guide, v1.11, August 2021. URL: https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp.