

# Faster amortized bootstrapping using the incomplete NTT for free

Thales B. Paiva<sup>1</sup>, Gabrielle De Micheli<sup>1</sup>, Syed Mahbub Hafiz<sup>1</sup>  
Marcos A. Simplicio Jr.<sup>1,2</sup> and Bahattin Yildiz<sup>1</sup>

<sup>1</sup> Future Security Team, LG Electronics USA, Inc.

<sup>2</sup> Universidade de São Paulo, São Paulo, Brazil

{thales.paiva,gabrielle.demicheli,syedmahbub.hafiz,bahattin.yildiz}@lge.com  
msimplicio@larc.usp.br

**Abstract.** Amortized bootstrapping techniques have been proposed for FHEW/TFHE to efficiently refresh multiple ciphertexts simultaneously within a polynomial modulus. Although recent proposals have very efficient asymptotic complexity, reducing the amortized cost essentially to  $\tilde{O}(1)$  FHE multiplications, the practicality of such algorithms still suffers from substantial overhead and high decryption failure rates (DFR). In this study, we improve upon one of the state-of-the-art amortized bootstrapping algorithms (Guimarães et al., ASIACRYPT 2023) for FHEW/TFHE-like schemes by introducing an alternative algorithmic strategy. Specifically, we combine Guimarães et al.’s strategy based on a two-part NTT with an incomplete Number Theoretic Transform (NTT) algorithm. The resulting construction is such that the multiplication of higher-degree polynomials that would usually create a bottleneck in an incomplete NTT setting actually comes for free. As a result, we demonstrate a  $2.12\times$  speedup compared to the algorithm of Guimarães et al. and a  $1.12\times$  improvement over the state-of-the-art (sequential) TFHE-rs while achieving a DFR close to  $2^{-32}$  for 7-bit messages, although the DFR is higher for 8-bit messages. We also explore trade-offs between execution time and DFR, identifying parameter sets that improve the execution time of Guimarães et al. by  $1.41\times$ , while simultaneously reducing the DFR by a factor of  $2^{-22}$  for 8-bit messages.

**Keywords:** Fully homomorphic encryption · FHEW/TFHE · amortized bootstrapping · incomplete NTT · efficient implementation

## 1 Introduction

Fully homomorphic encryption (FHE) was first introduced by Gentry in [Gen09] and has since been extensively studied and improved both in terms of efficiency and security. Although most modern FHE schemes are based on the Ring Learning With Errors (RingLWE) problem [SSTX09, LPR10], different strategies have been adopted in their construction. In schemes such as BGV/BFV [BGV14, Bra12, FV12], ciphertexts encrypt vectors of values, while operations on ciphertexts (e.g., addition and multiplication) are performed component-wise using ring operations. Such schemes benefit from an amortized operation cost by simultaneously performing computations on all the entries of the ciphertext vector. However, this comes at the cost of an expensive bootstrapping procedure and large parameters. Such large parameters also imply the need for stronger security assumptions, like relying on worst-case lattice-based problems with super-polynomial approximation factors.

In contrast, schemes such as FHEW/TFHE [DM15, CGGI20] operate at the bit level, and ciphertexts are LWE encryptions without additional ring structure. Nonetheless, the ring structure can still be explored in the context of programmable bootstrapping, where

an encryption of  $f(m)$  is computed for any function  $f$  and message  $m$ . By combining homomorphic LWE addition and programmable bootstrapping, FHEW/TFHE can evaluate any given circuit, i.e., perform arbitrary homomorphic computations. The resulting schemes benefit from simpler and, in principle, faster bootstrapping algorithms than in BGV/BFV. Moreover, they use much smaller parameters, allowing for (weaker) security assumptions based on worst-case lattice-based problems with polynomial approximation factors – similarly to the assumptions found in general lattice-based public key encryption schemes.

However, when closely analyzing the bootstrapping procedures of FHEW/TFHE and BGV/BFV, which are the main bottlenecks in those schemes, the latter has one clear perk: the amortization resulting from packing several messages into a large ring ciphertext greatly outweighs its higher execution costs. In light of this observation, amortized bootstrapping algorithms were also proposed to refresh multiple FHEW/TFHE ciphertexts at once. The ultimate goal of such proposals is to bridge the gap between the BGV/BFV and FHEW/TFHE strategies, while preserving the latter’s weaker security assumptions. Our work follows this trend, contributing to a line of investigation that attempts to answer the following research question: *can we bring amortized bootstrapping algorithms to be as efficient as (or more efficient than) their sequential counterparts?*

**Related Works.** The first amortized method for FHEW-like ciphertexts was proposed by Micciancio and Sorrell in [MS18]. The work shows that the cost of bootstrapping  $n$  ciphertexts can be reduced from  $\tilde{O}(n)$  cryptographic operations to  $\tilde{O}(n^\epsilon)$ , for any  $\epsilon > 0$ . The main idea of their method is to pack  $n$  LWE input ciphertexts into a single RingLWE ciphertext, and then perform the bootstrapping operation over the latter to amortize the overall cost over  $n$  messages. The main bottleneck of the resulting bootstrapping procedure is that it requires a homomorphic polynomial multiplication. Since FHEW [DM15] uses RingGSW registers to encrypt messages  $m$  in the exponent as monomials  $X^m$ , homomorphic addition is rather straightforward, as it translates to ciphertext multiplication. However, other homomorphic operations required by fast polynomial multiplication algorithms, such as subtraction or multiplication by so-called “twiddle factors”, become much more challenging. To address this issue, the original algorithm employs (a variant of) the Nussbaumer Transform [Nus80] to perform fast polynomial multiplication. Although the proposed method offers a major asymptotic performance improvement, the algorithm is hardly practical due to a large constant  $2^{O(1/\epsilon)}$  hidden in the asymptotic notation.

In subsequent works [GPVL23, DMKMS24], the Nussbaumer Transform is replaced with the much more practical Number Theoretic Transform (NTT), where the multiplication by twiddle factors is done via exponentiations, taking advantage of ring automorphisms. This approach for homomorphic multiplication is not new, as it was used in FHEW-related algorithms in [BDF18, LMK<sup>+</sup>23], for example. The result is that [GPVL23, DMKMS24] provide a much smaller overhead than [MS18], effectively reducing the underlying costs from  $2^{O(1/\epsilon)} \cdot n^{1/\epsilon}$  to  $(1/\epsilon) \cdot n^{1/\epsilon}$ . Although both algorithms follow the same structure, [DMKMS24] centers its algorithm around RingLWE registers to perform exponentiations, while [GPVL23] extends the automorphism multiplication technique to work directly on RingGSW ciphertexts. Moreover, the latter work proposes a concrete C++ implementation of their algorithm, which is the main target of our proposed improvements.

In another recent result, Liu and Wang [LW23a, LW23b] introduced a new algebraic framework for batch homomorphic computation relying on the tensoring of three rings. This alternative framework improves the efficiency of amortized bootstrapping algorithms, using  $\tilde{O}(1)$  homomorphic multiplications while still considering a polynomial modulus. While the asymptotic cost is (almost) optimal, it is shown in [DMKMS24] that the batch algorithm presented in [LW23b] suffers from the same limitations as [MS18]: its actual overhead is very high, making the algorithm impractical for concrete parameters. To

overcome this issue while preserving the same asymptotic complexity as [LW23b], Liu and Wang [LW23c] proposed an alternative algorithm that is efficient for concrete parameters. Their strategy relies on a scheme-switching technique from FHEW/TFHE to BFV/BGV ciphertexts, whose drawback is the need for using superpolynomial modulus instead of the polynomial moduli commonly found in all aforementioned schemes.

**Contributions.** In this paper, we enhance the amortized bootstrapping approach by Guimarães et al. [GPVL23] by targeting its NTT-based homomorphic polynomial multiplication required for homomorphic decryption. Specifically, their algorithm operates in cyclotomic rings of the form  $\mathbb{Z}_p[X]/(X^N + 1)$ , where  $N$  is a power of two and  $p$  is a prime modulus. When  $p \equiv 1 \pmod{2N}$ , such rings are known to support fast NTT computations using FFT-like iterative algorithms. However, [GPVL23] observes that these cannot be used in a homomorphic setting because they lead to a large error growth, raising the scheme’s decryption failure rate (DFR). They then propose a two-part NTT to provide a good balance between performance and error growth.

In contrast, our proposal for homomorphic polynomial multiplication leverages the concept of incomplete NTT [LS19, ABD<sup>+</sup>21, Kan22, DMKMS24]. This requires a redesign of the homomorphic decryption algorithm to account for a more complex multiplication in the NTT domain, as well as a different two-part inverse NTT (INTT) algorithm. This multiplication of higher-degree polynomials that usually creates a bottleneck in an incomplete NTT setting actually comes *for free* in our novel construction. We can then benefit from one important property of an incomplete NTT: its implementation allows for relaxed constraints and, hence, supports a wider set of parameters. Specifically, while the complete NTT in cyclotomic rings of the form  $\mathbb{Z}_p[X]/(X^N + 1)$  requires  $p \equiv 1 \pmod{2N}$ , the  $\ell$ -incomplete NTT only requires the less restrictive condition  $p \equiv 1 \pmod{2N/2^\ell}$ . This flexibility allows using a smaller prime modulus  $p$ , which is one source of performance gains in the amortized bootstrapping algorithm we propose. At the same time, the broader choices of  $p$  allow us to explore new tradeoffs between DFR and performance.

In summary, our contributions are as follows:

1. *Algorithmic contribution.* We introduce an improved amortized bootstrapping algorithm that exploits incompleteness in homomorphic (inverse) NTT operations without incurring any additional overhead. Specifically, we propose an alternative two-part incomplete NTT that generalizes the two-part inverse NTT used in [GPVL23], while preserving the same algorithmic complexity. This generalization enables using a radix-2 incomplete NTT formulation, setting the stage for different performance gains and speed-DFR tradeoffs.
2. *Performance improvements and tradeoffs.* We explore different parameter settings and introduce concrete, optimized amortized bootstrapping instantiations that are faster than the state of the art. In particular, our experimental results show up to a 2.12× improvement in execution time compared to [GPVL23] and a 1.12× speedup compared to the state-of-the-art (sequential) TFHE-rs [Zam22] for a DFR close to  $2^{-32}$  for 7-bit messages, although the DFR is higher for 8-bit messages. In addition, we analyze the tradeoff between execution time and DFR in our algorithm and identify configurations where execution time improves by 1.41× over [GPVL23], while simultaneously reducing the DFR from  $2^{-17}$  to  $2^{-39}$  for 8-bit messages. Finally, we demonstrate that DFR can be reduced below that of TFHE-rs ( $2^{-131} < 2^{-128}$ ) at the cost of a 2.9× performance slowdown for 8-bit messages.
3. *Reproducible results with open-source implementation.* We provide a C++ implementation of our algorithm, built on top of the codebase from [GPVL23]. The code will be open-sourced upon acceptance of the paper.

*Remark 1.* We became aware of a very recent work [GP25], published on ePrint on April 15th, which also improves upon the amortized bootstrapping algorithm of Guimarães et al. [GPVL23], albeit using very different techniques. While our work optimizes the original NTT-based polynomial multiplication, [GP25] introduces a novel multiplication technique based on sparse polynomials. Due to the timing of its publication, we do not discuss [GP25] further in this version.

**Organization.** Section 2 reviews the necessary mathematical background and theory relevant to our work, including the concept of incomplete NTT and amortized bootstrapping. Section 3 gives an overview of the amortized bootstrapping strategy proposed in [GPVL23], discussing their two-part inverse NTT along with security considerations and implementation aspects. Section 4 details our bootstrapping algorithm, including our alternative two-part inverse incomplete NTT. Section 5 describes our implementation and experimental results, which are compared with state-of-the-art sequential and amortized approaches. Finally, Section 6 presents our conclusions and ideas for future work.

## 2 Preliminaries

Throughout this paper, we will write polynomials  $\mathbf{x} = \sum x_i X^i$  in *italic bold* to differentiate them from vectors  $\mathbf{v}$  that are written in **bold**. For a polynomial  $\mathbf{x}$  we will use both  $x_i$  to refer to its  $i^{\text{th}}$  coefficient or the notation  $\mathbf{x}[i]$  in algorithms' pseudo-code. For matrices, we will denote as  $\mathbf{X}[i]$  their  $i^{\text{th}}$  row. We use  $\lg$  to denote the logarithm in base 2.

### 2.1 Rings and schemes

#### 2.1.1 Rings

The algorithm given in [GPVL23] operates over a circulant ring of the form  $\mathbb{Z}[X]/(X^p - 1)$  for some prime  $p$ . Using the same notation as in [GPVL23], we will refer to this ring as  $\tilde{\mathcal{R}}$ . When the coefficients of the elements of the ring are taken modulo  $Q$  for any positive integer  $Q$ , then we will write  $\tilde{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^p - 1)$ . The algorithm (both in [GPVL23] and ours) will also rely on power-of-two cyclotomic rings of the form  $\mathbb{Z}[X]/(X^N + 1)$  where  $N = 2^k$  for some  $k \in \mathbb{N}$ . We will denote this ring  $\hat{\mathcal{R}}$  or  $\hat{\mathcal{R}}_Q$  if taken modulo  $Q$ .

#### 2.1.2 Standard encryption schemes

We recall definitions and notations for the standard LWE encryption scheme used in the bootstrapping algorithm. We also extend our description to the ring version of LWE, the circulant version of LWE and GSW, all used in the algorithm.

**LWE.** Consider some positive integers  $n$  and  $q$ . Let  $\mathbf{s} \leftarrow \chi$  be a secret key sampled from a distribution  $\chi$ ,  $m \in \mathbb{Z}$  a message, and  $\Delta \in \mathbb{Z}$  a scaling factor. The LWE encryption of the message  $m$  under the secret key  $\mathbf{s}$  is given by

$$\text{LWE}_{q,\mathbf{s}}(\Delta \cdot m) = [\mathbf{a}^\top, b] \in \mathbb{Z}_q^{1 \times (n+1)},$$

where  $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ ,  $b = -\mathbf{a} \cdot \mathbf{s} + e + \Delta \cdot m \in \mathbb{Z}_q$  and  $e \leftarrow \chi'$  is the error, sampled from a distribution  $\chi'$ , and ciphertexts are represented as *row* vectors.

**Ring LWE.** The ring version of LWE considers the ring  $\mathcal{R}_q = \mathbb{Z}_q[X]/(\Phi_m(X))$ , where  $\Phi_m$  is the  $m^{\text{th}}$  cyclotomic ring. Let  $\mathbf{sk} \leftarrow \chi$  be a secret key sampled from a distribution  $\chi$ ,

$\mathbf{m} \in \mathcal{R}_q$  a message and  $\Delta \in \mathbb{N}$  a scaling factor. The RingLWE encryption of the message  $\mathbf{m}$  under the secret key  $\mathbf{sk}$  is given by

$$\text{RingLWE}_{q,\mathbf{sk}}(\Delta \cdot \mathbf{m}) = [\mathbf{a}, \mathbf{b}] \in \mathcal{R}_q^{1 \times 2},$$

where  $\mathbf{a} \leftarrow \mathcal{R}_q$  and  $\mathbf{b} = -\mathbf{a} \cdot \mathbf{sk} + \mathbf{e} + \Delta \cdot \mathbf{m}$  and  $e_i \leftarrow \chi'$  for each coefficient  $e_i$  of the error. In order to decrypt a RingLWE ciphertext, one computes the quantity  $\mathbf{b} + \mathbf{a} \cdot \mathbf{sk} = \mathbf{e} + \Delta \cdot \mathbf{m}$  and performs a rounding operation to recover the message  $\mathbf{m}$ . Most schemes, including the bootstrapping algorithms considered in this work, consider  $\Phi_m(X) = X^N + 1$  for  $N = 2^k$  and any  $k \in \mathbb{N}$ , and thus the RingLWE problem is built with  $\hat{\mathcal{R}}$ .

**Circulant LWE.** As mentioned above, the algorithm introduced in [GPVL23] relies on circulant rings and thus considers the hardness of a variant of LWE called circulant-LWE (CLWE). This variant was introduced in [BDF18] and proven to be as hard as the RingLWE variant on prime-order cyclotomic polynomials. It essentially considers the ring  $\mathcal{R}$  instead of  $\hat{\mathcal{R}}$ , and samples are obtained by considering RingLWE samples and projecting them onto  $\mathcal{R}$ .

**GSW.** Consider a modulus  $q$  and a dimension parameter  $N$ . The GSW scheme encrypts a message  $\mathbf{m}$  when  $\mathbf{C} \cdot \mathbf{v} = \mathbf{m} \cdot \mathbf{v} + \mathbf{e}$ , where the ciphertext  $\mathbf{C}$  is an  $N \times N$  matrix over  $\mathbb{Z}_q$ , the secret key  $\mathbf{v}$  is  $N$ -dimensional vector over  $\mathbb{Z}_q$  and  $\mathbf{e}$  is a small error vector [GSW13]. In [GPVL23], the authors consider a double-CRT version of GSW, motivated by the fact that the modulus considered is quite large. More precisely, the modulus  $Q$  is decomposed into a product of smaller primes  $q_i$ , and CRT is used to operate independently modulo each  $q_i$ . Moreover, the polynomials in the ciphertexts are all represented in the FFT domain, thus the name double-CRT or RNS representation (see [GPVL23, Section 2.5] for more details).

## 2.2 Standard operations

**Modulus switching.** Modulus switching is a standard operation that changes the modulus of a ciphertext. For RingLWE ciphertext, the operation takes  $(\mathbf{a}, \mathbf{b}) = \text{RingLWE}_{q,\mathbf{sk}}(\Delta \cdot \mathbf{m})$  and outputs a ciphertext  $(\mathbf{a}', \mathbf{b}') = \text{RingLWE}_{q',\mathbf{sk}}(\Delta \cdot \mathbf{m})$  for a new (usually smaller) modulus  $q'$ . We refer to [GPVL23, Algorithm 2] for the pseudo-code of the modulus switching algorithm. In their bootstrapping algorithm, the modulus switching operation is executed on RingLWE ciphertexts only.

**Key switching.** Key switching is a standard operation that allows to switch of the secret key from  $\mathbf{sk}$  to another key  $\mathbf{zk}$ . More precisely, in the RingLWE case, it takes a ciphertext  $(\mathbf{a}, \mathbf{b}) = \text{RingLWE}_{q,\mathbf{sk}}(\Delta \cdot \mathbf{m})$  and a public key-switching key generated from the secret key  $\mathbf{sk}$  and outputs a ciphertext  $(\mathbf{a}', \mathbf{b}') = \text{RingLWE}_{q,\mathbf{zk}}(\Delta \cdot \mathbf{m})$ . The specific case of circulant rings defined modulo  $X^p - 1$  can pose some security issues. Indeed, as explained in [GPVL23], it was proven in [BDF18] that the GSW scheme over such circulant rings is secure when only encryptions of powers of  $X$  are considered. However, in key switching, one must consider the encryption of the new secret key  $\mathbf{zk}$ , which is not a power of  $X$ . To address this, [GPVL23, Section 3.5] introduces two algorithms specifically for GSW key switching with a noise-memory tradeoff. We refer the reader to [GPVL23, Algorithms 5 and 6] for details about their algorithms.

**Automorphisms.** Ring automorphisms can be used to perform scalar multiplication with registers. Recall that an automorphism is a bijective map from a ring  $\mathcal{R}$  to itself such that for a given  $t \in \mathbb{Z}_q^*$ , we have  $\mathbf{a}(X) \mapsto \mathbf{a}(X^t)$ . Both [LMK<sup>+</sup>23] and [DMKMS24] apply

ring automorphisms on RingLWE ciphertexts. In [GPVL23, Section 3.6], automorphisms are defined over GSW ciphertexts. Consider the GSW ciphertext  $\mathbf{C}$  (a matrix) defined over the ring  $\tilde{\mathcal{R}}_Q$ , which encrypts a given message  $\mathbf{m}$  under a certain key  $\mathbf{sk}$ . We also consider a switching key  $\mathbf{ak}_t$ . An automorphism  $\psi_t : \tilde{\mathcal{R}}_Q \rightarrow \tilde{\mathcal{R}}_Q$  can be applied to each row of  $\mathbf{C}$  such that  $\psi_t(\mathbf{C}) \in \tilde{\mathcal{R}}_Q \text{GSW}_{\psi_t(\mathbf{sk})}(\psi_t(\mathbf{m}))$ . Applying key switching with  $\mathbf{ak}_t$  on  $\psi_t(\mathbf{C})$  allows to recover a ciphertext  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_{\mathbf{sk}}(\psi_t(\mathbf{m}))$ .

### 2.3 Incomplete NTT

The number-theoretic transform (NTT) is a finite-field analogue of the fast Fourier transform (FFT) that can be used for fast polynomial multiplication under certain conditions. For negacyclic rings of the form  $\mathbb{Z}_p[X]/(X^N + 1)$ , NTT-based multiplication requires the existence of a  $2N$ -root of unity in  $\mathbb{Z}_p$ , which is guaranteed to exist when  $p \equiv 1 \pmod{2N}$ . Similar to the FFT, the NTT is efficiently computed using  $O(N \log N)$  operations by an iterative process. Because of the strict requirements on the modulus  $p$  from using the NTT in  $\mathbb{Z}_p[X]/(X^N + 1)$ , many works consider the so-called  $\ell$ -incomplete NTT formulation [ABD<sup>+</sup>19, CHK<sup>+</sup>21, Lip21, ABC19, CP05, Moe76, DMKMS24]. This is a generalization of the NTT where, instead of running all the  $\lg N$  layers, one stops the iterations at  $\lg N - \ell$  layers, for an integer  $0 \leq \ell \leq \lg N$ . This requires a  $2N/2^\ell$ -root of unity to exist in  $\mathbb{Z}_p$ , which relaxes the condition on the modulus to  $p \equiv 1 \pmod{2N/2^\ell}$ .

Due to the properties of cyclotomic polynomials, when a  $2N/2^\ell$ -root of unity exists in  $\mathbb{Z}_p$ , the polynomial  $X^N + 1$  can be factored as  $X^N + 1 = (X^{2^\ell} - \psi_1) \dots (X^{2^\ell} - \psi_{N/2^\ell})$ , where each  $\psi_i$  is a different power of the root of unity that can be easily computed. Then, for a given polynomial  $\mathbf{a} \in \mathbb{Z}_p[X]/(X^N + 1)$ , the  $\ell$ -incomplete NTT returns a sequence of  $N/2^\ell$  polynomials defined as  $\text{NTT}(\mathbf{a}) = (\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_{N/2^\ell})$ , where each  $\hat{\mathbf{a}}_i = \mathbf{a} \bmod X^N - \psi_i$ . To compute the product  $\mathbf{c} = \mathbf{a}\mathbf{b}$  of two polynomials  $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p[X]/(X^N + 1)$  using the  $\ell$ -incomplete NTT, one starts by computing  $\hat{\mathbf{a}} = \text{NTT}(\mathbf{a})$  and  $\hat{\mathbf{b}} = \text{NTT}(\mathbf{b})$ . Then, one computes the products  $\hat{\mathbf{c}}_i = \hat{\mathbf{a}}_i \hat{\mathbf{b}}_i \bmod X^N - \psi_i$  for  $i = 1, \dots, N/2^\ell$ . The process of multiplying polynomials modulo  $X^N - \psi_i$  for each  $\psi_i$ , which is a frequent operation in the NTT formulation, is usually referred to as the base multiplication. Finally, the product  $\mathbf{c}$  is computed by taking the inverse incomplete NTT as  $\mathbf{c} = \text{INTT}(\hat{\mathbf{c}})$ , where  $\hat{\mathbf{c}} = (\hat{\mathbf{c}}_1, \dots, \hat{\mathbf{c}}_{N/2^\ell})$ .

The  $\ell$ -incomplete NTT is known to enable faster polynomial multiplication than the complete NTT (which corresponds to the case where  $\ell = 0$ ) in some cases, typically when using small values of  $\ell$  such as  $\ell = 1, 2$  [ABD<sup>+</sup>21, Kan22]. However, in general, increasing  $\ell$  beyond these values degrades performance as the complexity of the base multiplication, typically executed using the schoolbook algorithm, grows exponentially in  $\ell$ .

## 3 Guimarães et al.'s [GPVL23] amortized bootstrapping

The amortized bootstrapping algorithm presented in [GPVL23] follows the same structure as the original amortized bootstrapping algorithm introduced in [MS18], that is, it can be decomposed into three main steps:

1. **Packing** takes as input  $N$  LWE ciphertexts, each encrypting a (scaled)  $\kappa$ -bit message  $m_i \in \mathbb{Z}_{2^\kappa}$  under the same secret key  $\mathbf{s}$ , i.e., we consider a set  $\{\text{LWE}_{\mathbf{s}}^{p^*}(\Delta \cdot m_i)\}_{i=1}^N$  for prime modulus  $p^*$  and a scaling factor  $\Delta = \lfloor p^*/2^\kappa \rfloor$ , a packing key consisting of RingLWE encryptions of coefficients of the LWE secret key and outputs a single “packed” RingLWE ciphertext encrypting a (scaled) message  $\mathbf{m}(X) = \sum_i m_i X^{i-1}$ . The algorithm was introduced in [MS18, Section 3] and we refer to [MS18, Lemma 4] for a proof of its correctness and the corresponding pseudocode. In [GPVL23],

the packing step is not modified from the original work, and the authors report the pseudocode in Appendix E.

2. **Homomorphic decryption** is the core of the algorithm and corresponds to the bootstrapping step. Homomorphically decrypting the RingLWE ciphertext obtained from packing essentially consists of computing a homomorphic polynomial multiplication as explained in Section 2.1.
3. **Message extraction** is the last step of the amortized bootstrapping algorithm, where one recovers from the bootstrapped ring ciphertext the  $N$  LWE ciphertexts with reduced noise. This technique was introduced in [DM15] and adapted to this algorithm (in particular to the cyclic ring setup) in [GPVL23, Algorithm 10].

Let us now focus on the second step, i.e., the homomorphic decryption. The output of the previous step, the packing algorithm, is a RingLWE ciphertext  $(\mathbf{a}, \mathbf{b}) \in \hat{\mathcal{R}}_Q^2$ . Recall from Section 2.1 that in order to (homomorphically) decrypt such a ciphertext encrypted under some key  $\mathbf{zk}$ , one must compute the quantity  $\mathbf{b} + \mathbf{a} \cdot \mathbf{zk}$  (with an encryption of  $\mathbf{zk}$  given as bootstrapping key). For the polynomial multiplication, [GPVL23] replaces the Nussbaumer transform and the SlowMult algorithm from [MS18] by an NTT and a pointwise multiplication. However, in order for this substitution to work, the algorithm must rely on an accumulator that enables the evaluation of a standard NTT homomorphically. This includes being able to perform operations in the exponent such as addition, subtraction, scalar multiplication, and key-switching.

The algorithm introduced in [GPVL23] primarily operates over the circulant ring  $\tilde{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^p - 1)$  for some prime  $p$  and modulus  $Q$  and the registers on which the homomorphic operations are performed are RingGSW ciphertexts over this ring encrypting with the secret key  $\mathbf{sk}$ . While homomorphic multiplications and external products are common operations already well-defined for (ring) GSW, [GPVL23] extends the use of automorphisms and key switching to the GSW scheme [GPVL23, Section 3.5, Algorithm 5] to enable the use of the homomorphic NTT. Now, as the inputs of the polynomial multiplication are elements  $\mathbf{a}, \mathbf{zk} \in \hat{\mathcal{R}}_Q$ , a negacyclic polynomial ring, the multiplication (and addition of  $\mathbf{b} \in \hat{\mathcal{R}}_Q$ ) is done by computing

$$\text{INTT}(\text{NTT}(\mathbf{a}) \cdot \text{NTT}(\mathbf{zk}) + \text{NTT}(\mathbf{b})) \equiv \mathbf{a} \cdot \mathbf{zk} + \mathbf{b} \in \hat{\mathcal{R}}_Q.$$

The quantities  $\text{NTT}(\mathbf{a})$  and  $\text{NTT}(\mathbf{b})$  are first computed along with the bootstrapping keys which correspond to RingGSW encryptions of  $\text{NTT}(\mathbf{zk})$ . The pointwise multiplication is performed using automorphisms. Finally, the inverse NTT is applied homomorphically in order to recover the GSW ciphertexts encrypting  $\mathbf{b} + \mathbf{a} \cdot \mathbf{zk}$ . The correctness of the amortized bootstrapping algorithm is given in [GPVL23, Theorem 1] and the noise growth is reported in [GPVL23, Corollary 2]. We report in Table 1 the parameters used in both [GPVL23] and our algorithm.

In the following section, we describe the inverse NTT algorithm used by Guimarães et al. [GPVL23] in more details. We note that the presentation is adapted to better suit the comparison with our own work.

### 3.1 Two-part inverse NTT for controlling error growth

Consider the most costly operation when bootstrapping a ciphertext  $(\mathbf{a}, \mathbf{b}) \in \hat{\mathcal{R}}_Q$  encrypted with a secret key  $\mathbf{zk}$ , which is the homomorphic multiplication of the polynomials  $\mathbf{a}$  and  $\mathbf{zk}$  given an encryption of the coefficients of  $\mathbf{zk}$ . Usually, multiplications in the cyclotomic ring  $\hat{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$  can be done very efficiently in  $O(N \lg N)$  using the fast inverse NTT (INTT), which is analogous to the in-place inverse FFT.

However, fast INTT can be problematic when using homomorphic computations due to the resulting error growth. Indeed, at each level  $i$ , for  $i = 1$  to  $\lg N$ , the algorithm

**Table 1.** Summary of the parameters used in [GPVL23] and our algorithm.

Type	Parameter	Description
Message	$\kappa$	Bitsize of the messages
Degree	$N$	Power-of-two degree of RLWE samples
	$p$	Prime degree of circulant-LWE samples
Modulus	$Q$	Ciphertext modulus
	$p^*$	Plaintext modulus
Rings	$\hat{\mathcal{R}} = \mathbb{Z}[X]/(X^N + 1)$	Power-of-two cyclotomic ring
	$\hat{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$	Power-of-two cyclotomic ring with coefficients modulo $Q$
	$\tilde{\mathcal{R}} = \mathbb{Z}[X]/(X^p - 1)$	Circulant ring
	$\tilde{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^p - 1)$	Circulant ring with coefficients modulo $Q$
Secret Key	$\mathbf{s}$	LWE secret key
	$\mathbf{sk}$	RingGSW secret key in $\tilde{\mathcal{R}}_Q$ .
	$\mathbf{zk}$	RingLWE secret key in $\hat{\mathcal{R}}_Q$
NTT-related	$m$	Radix of NTT
	$\ell$	NTT incompleteness degree
	$\alpha$	Balancing parameter for the two-part NTT

performs  $N/2$  multiplications and  $N$  additions. While this would require only  $(N/2) \lg N$  multiplications for cleartext operations, in the homomorphic case each INTT level requires scalar multiplications on ciphertexts, thus increasing the depth of multiplications. With resulting ciphertexts having a depth of  $\lg N$  multiplications, their decryption will likely fail due to error growth except for very small values of  $N$ .

Let us compare this to the naive INTT computation using a matrix-vector multiplication. The inverse NTT is a linear operation and thus can be described with a matrix  $\mathbf{M} \in \mathbb{Z}_p^{N \times N}$ , so that the INTT computation now takes  $N^2$  multiplications. In this case, all homomorphic multiplications are done in depth 1, and then combined with additions, making it much more manageable with respect to ciphertext error growth. However, on the downside, the large number of multiplications required can make it impractical for larger values of  $N$ .

The approach taken by Guimarães et al. [GPVL23] is to find a good balance between performance and depth of homomorphic multiplications (with respect to error growth). In particular, they use a two-part INTT computation based on a radix- $m$  formulation of the INTT, which limits the homomorphic multiplication depth to only two levels. Algorithm 1 shows their INTT algorithm without considering homomorphic operations. It takes an input<sup>1</sup>  $0 \leq \alpha \leq \lg N$  that defines the radix  $m = 2^\alpha$  used for the computation using  $N^2/m + Nm = N^2/2^\alpha + N2^\alpha$  operations. After a simple optimization argument, the authors conclude that using  $\alpha \approx \lg N/2$ , the value that best balances Part 1 and Part 2 in Algorithm 1, is the theoretically optimal choice.

However, in practice, there are cases where another choice of  $\alpha$  may yield better performance. This can be explained by making the following observations. In efficient FHE implementations, it is not uncommon to merge steps between different procedures that are run sequentially to improve performance and error growth. This is done in Guimarães et al.'s implementation for example as they partially merge the homomorphic inverse NTT with the message extraction step to reduce the noise growth [GPVL23, Algorithm 10 (see Remark 1)]. Moreover, to optimize performance, one may prefer to use a more efficient (but with higher noise growth) scalar multiplication or a slower (but more precise) multiplication in different parts of the code. This can also be seen in Guimarães et al.'s implementation where either scalar multiplications with RGSW or RLWE registers are

<sup>1</sup>The algorithm by Guimarães et al. [GPVL23] takes parameter  $m$  directly. We introduce the auxiliary variable  $\alpha = \lg m$  to make our algorithm (described in Section 4.1) more easily comparable to theirs.

```

1: procedure INTT_gpv1( $\hat{\mathbf{a}}, \alpha$ )
  ▶  $\hat{\mathbf{a}} \in \mathbb{Z}_p^N$  is the vector of coefficients to which the INTT is applied, i.e.,  $\hat{\mathbf{a}} = \text{NTT}(\mathbf{a})$ 
  ▶  $\alpha \in \mathbb{Z}$  is the balancing parameter between parts 1 and 2
2:    $\omega \leftarrow$  a primitive  $2N$ -th root of unity of  $p$ 
3:    $m = 2^\alpha$  ▷ Defines the radix- $m$  INTT computation
4:   ▷ Part 1:  $O(m(N/m)^2) = O(N^2/2^\alpha)$ 
5:   for  $k = 0$  to  $m - 1$  do
6:      $\mathbf{g} \leftarrow (\hat{\mathbf{a}}[k], \hat{\mathbf{a}}[k + m], \dots, \hat{\mathbf{a}}[N - 2m], \hat{\mathbf{a}}[N - m])$ 
7:      $\mathbf{h}_k \leftarrow \mathbf{0} \in \mathbb{Z}_p^{N/m}$ 
8:     for  $j = 0$  to  $N/m - 1$  do
9:        $\mathbf{h}_k[j] \leftarrow \sum_{i=0}^{N/m-1} \mathbf{g}[i] \omega^{-2ij(N/m)} \pmod p$ 
10:    ▷ Part 2:  $O(m^2(N/m)) = O(N2^\alpha)$ 
11:    for  $k_1 = 0$  to  $N/m - 1$  do
12:      for  $k_2 = 0$  to  $m - 1$  do
13:         $j \leftarrow k_1 + k_2(N/m)$ 
14:         $\mathbf{a}[j] \leftarrow \frac{1}{N} \sum_{i=0}^{m-1} \mathbf{h}_i[k_1] \omega^{-j-2i(k_1-k_2(N/m))} \pmod p$ 
15:    ▷ Notice that the output is in natural, not bit-reversed, order
16:    return  $\mathbf{a} \in \mathbb{Z}_p^N$ 
17: end procedure

```

**Algorithm 1.** Two-part inverse NTT computation used by Guimarães et al. [GPVL23].

considered in each parts of Algorithm 1.

Such practical considerations may result in some imbalance in the performance of the two parts of Algorithm 1, making the choice of  $\alpha$  different than the theoretical optimum. This leads to different ways of computing sums of products in Part 1 and Part 2 of Algorithm 1 and, to the best of our knowledge, explains why they use  $\alpha = 6$  (i.e.  $m = 64$ ) for  $N = 1024$  in their code, whereas the theoretically best value of  $\alpha$  would be  $\alpha = 5$  if INTT was individually considered.

### 3.2 Secure and reliable instantiations

The security of Guimarães et al.'s [GPVL23] algorithm is based on the hardness of LWE under two different settings: one related to the LWE problem used for encryption of the original messages, and another for the RingLWE problem used to define the GSW-based accumulator. These are explicitly defined as follows.

1. For a prime modulus  $p^*$  and a power-of-two  $N$ , the first LWE setting is defined over the vector space  $\mathbb{Z}_{p^*}^N$ . The secret key  $\mathbf{s} \in \mathbb{Z}_{p^*}^N$  and the error vector  $\mathbf{e} \in \mathbb{Z}_{p^*}^N$  are sampled according to distributions  $\chi$  and  $\chi'$ , where  $\chi$  is a balanced ternary distribution<sup>2</sup> of even Hamming weight  $w$ , and  $\chi'$  is a discrete Gaussian with standard deviation of 1.
2. The second setting is a RingLWE defined over the circulant polynomial ring  $\tilde{\mathcal{R}}_Q = \mathbb{Z}_Q[X]/(X^p - 1)$ . The secret key  $\mathbf{sk} \in \tilde{\mathcal{R}}_Q$  and the error polynomial  $\mathbf{e} \in \tilde{\mathcal{R}}_Q$  are sampled according to discrete Gaussians of standard deviations of 3.2 and 1, respectively.

<sup>2</sup>A balanced ternary distribution of even Hamming weight  $w$  and length  $N$  samples vectors in  $\{-1, 0, 1\}^N$  whose numbers of positive and negative coordinates are equal to  $w/2$ .

While parameters  $(N, p^*, w, p, Q)$  used above define the security of the scheme, for it to be practical, the decryption failure rate (DFR) must be sufficiently small. Naturally, the DFR also depends on the message space  $\mathbb{Z}_{2^\kappa}$ , which affects the scaling factor used when encoding coefficients of  $\kappa$ -bit messages into  $\mathbb{Z}_{p^*}$ . Guimarães et al. [GPVL23] show that the DFR can be upper bounded by

$$\Pr(\text{Decryption of } \kappa\text{-bit message fails}) \leq 1 - \operatorname{erf}\left(\frac{p/2^{\kappa+1}}{\sqrt{2\varepsilon}}\right), \quad (1)$$

where  $\operatorname{erf}$  is the Gaussian error function and  $\varepsilon = (p + N^2/2)(\lg p^*)(p/p^*)^2 + w/12$ .

Notice that the authors provide a generic formulation of the upper bound that takes into account different parameter formulations (mainly standard deviations for LWE noise that are later set to 1). However, we chose the simpler and more concrete presentation above that both matches their results and is enough for our case. Then, to select parameters achieving 128-bit security and low DFR, Guimarães et al. [GPVL23] rely on the LWE estimator [APS15] and used a manual search. We report their parameter choices and performance in Section 5 Table 2 when comparing to our own results.

### 3.3 Acceleration using the Intel HEXL library

The code provided by Guimarães et al. [GPVL23] uses the Intel HEXL library [BKS<sup>+</sup>21] to speedup some of the mathematical computations. While HEXL comes with efficient vectorized implementations for number theory-related functions and multiplications in polynomial rings, they make some necessary adjustments to their code to be able to use HEXL for their purposes. For example, some of the most costly operations during bootstrapping take place inside the ring  $\tilde{\mathcal{R}} = \mathbb{Z}[X]/(X^p - 1)$ , but HEXL is optimized for fast NTT-based multiplication over rings of the form  $\tilde{\mathcal{R}}' = \mathbb{Z}[X]/(X^{N'} - 1)$ , where  $N'$  is a power of two (recall that  $p$  is a prime).

In order to use HEXL for operations in  $\tilde{\mathcal{R}}$ , they then notice that a product  $\mathbf{a} \cdot \mathbf{b}$  of polynomials  $\mathbf{a}, \mathbf{b} \in \tilde{\mathcal{R}}$  has degree at most  $2p - 2$ . Hence, if they consider  $N'$  as the first power of two larger than  $2p - 2$ , and represent the polynomials  $\mathbf{a}$  and  $\mathbf{b}$  as elements in  $\tilde{\mathcal{R}}'$ , the resulting product in  $\tilde{\mathcal{R}}'$  would be efficiently computed in HEXL and, because  $N'$  is larger than the maximum degree of the product, the result is not affected by the modulo  $(X^{N'} - 1)$ . Therefore, by manually reducing the result by  $(X^p - 1)$ , they can obtain the desired product  $\mathbf{a} \cdot \mathbf{b} \in \tilde{\mathcal{R}}$ .

The drawback of this approach is that memory usage and element-wise operations take roughly  $2\times$  more when operating in  $\tilde{\mathcal{R}}'$  instead of  $\tilde{\mathcal{R}}$  directly. They conclude that one should choose  $p$  as close to the next power of two as possible, which can be explained for two reasons. First, the bootstrapping performance is very similar for values of  $p$  with the same number of bits. Second, the LWE hardness and DFR are better when  $p$  is larger.

## 4 Amortized bootstrapping using the incomplete NTT

In this section, we describe how to use the incomplete NTT in the context of Guimarães et al.'s amortized bootstrapping algorithm. First, we show how to define a two-part incomplete NTT whose error growth is exactly the same as that obtained by Guimarães et al. [GPVL23]. We then show how the base multiplication can also be integrated into the formulation without impacting the error growth. Finally, we present a surprising property of our construction: it incurs no performance penalty when using moderate levels of incompleteness, which we refer to as *incompleteness for free*.

#### 4.1 An incomplete formulation of the two-part inverse NTT

The main idea is to use a similar approach to the two-part inverse NTT used by Guimarães et al., but generalizing it to enable the use of the radix-2 incomplete NTT formulation. One challenge is that the twiddle factors are not easily defined by a closed expression in this case because there is no guarantee that a primitive  $2N$ -th root of unity exists in  $\mathbb{Z}_p$ .

An easy way to compute the twiddle factors is by considering the factorization of the inverse NTT into matrices, analogously to how Gentleman and Sande [GS66] considered for the FFT. If we let  $\mathbf{M} \in \mathbb{Z}_p^{N \times N}$  be the matrix associated with the inverse NTT operation over bit-reversed vectors, then the well-known fast inverse incomplete NTT [LS19, Kan22] yields a factorization of  $\mathbf{M}$  into

$$\mathbf{M} = 2^{-(\lg N - \ell)} \mathbf{M}_1 \dots \mathbf{M}_{(\lg N - \ell)},$$

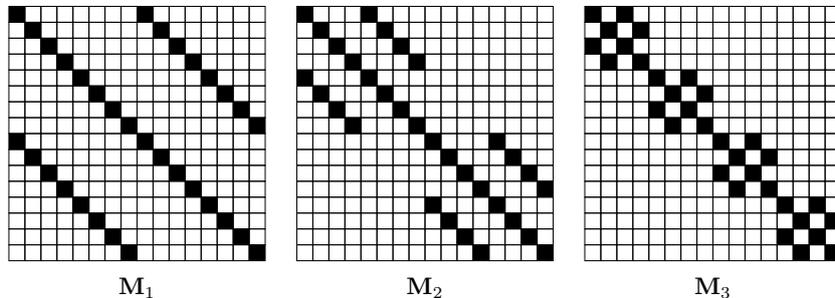
where  $\ell$  is the incompleteness level, and each  $\mathbf{M}_i$  is a sparse matrix, often referred to as a butterfly matrix. Figure 1 shows the overall shape of the butterfly matrices for the particular case when  $N = 16$  and  $\ell = 1$ .

Let  $\alpha$  be an integer such that  $0 \leq \alpha \leq \lg N - \ell$ . We can factor  $\mathbf{M}$  into two matrices as  $\mathbf{M} = \mathbf{M}_B \mathbf{M}_A$ , where  $\mathbf{M}_B = 2^{-(\lg N - \ell)} \mathbf{M}_1 \dots \mathbf{M}_\alpha$  and  $\mathbf{M}_A = \mathbf{M}_{\alpha+1} \dots \mathbf{M}_{(\lg N - \ell)}$ . Since  $\mathbf{M}_A$  and  $\mathbf{M}_B$  are products of the sparse butterfly matrices, they are somewhat sparse: each row of  $\mathbf{M}_B$  has  $2^\alpha$  nonzero coefficients and  $\mathbf{M}_A$  is a block diagonal matrix in which each block is an  $(N/2^\alpha) \times (N/2^\alpha)$  matrix. Figure 2 shows the sparsity of  $\mathbf{M}_A$  and  $\mathbf{M}_B$  for  $N = 16$ ,  $\ell = 1$  and  $\alpha = 2$ .

Algorithm 2 shows how the sparsity of the matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$  can be used to compute the incomplete NTT in two steps: the multiplications required by both parts 1 and part 2 are implemented by discarding the null entries of matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$ , respectively. Notice that, since  $N$  and  $p$  are part of the scheme parameters, we can easily precompute matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$  for the targeted incompleteness level  $\ell$ . With these precomputed matrices, Algorithm 2 uses  $O(N2^\alpha + N^2/2^\alpha)$  operations and hence has the same complexity as Algorithm 1.

**Generalizing to three-part inverse NTT or more.** We remark that it is possible to split the inverse NTT into more than only two parts by grouping more products of butterfly matrices that factor  $\mathbf{M}$ . This is analogous to using a larger number of radix- $m$  splitting layers as considered by Guimarães et al. [GPVL23]. While this would make the inverse NTT computation more efficient<sup>3</sup>, their conclusion also holds for our formulation: if we

<sup>3</sup>To see why more splittings make the inverse NTT more efficient, remember that the fastest way to compute the INTT is to split it into  $\lg N$  parts, each corresponding to one butterfly matrix.



**Figure 1.** The shape of the butterfly matrices that decompose the inverse NTT for  $N = 16$  and  $\ell = 1$ . White and black squares represent the positions of null and non-null values, respectively.

```

1: procedure INTT_incomplete( $\hat{\mathbf{a}}, \ell, \alpha$ )
  ▶  $\hat{\mathbf{a}} \in \mathbb{Z}_p^N$  is the vector of coefficients to which the INTT is applied, i.e.,  $\hat{\mathbf{a}} = \text{NTT}(\mathbf{a})$ 
  ▶  $\ell \in \mathbb{Z}$  is the incompleteness level (compatible with  $N$  and  $p$ )
  ▶  $\alpha \in \mathbb{Z}$  is the balancing parameter between parts 1 and 2
2:   ▷ Build matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$  (can be precomputed for fixed  $(N, p, \ell, \alpha)$ )
3:    $\mathbf{M} \leftarrow$  Matrix associated with the  $\ell$ -incomplete INTT of length  $N$  in  $\mathbb{Z}_p$ 
4:    $\mathbf{M}_1, \dots, \mathbf{M}_{(\lg N - \ell)} \leftarrow$  Butterfly matrices that factor  $\mathbf{M}$ 
5:    $\mathbf{M}_A \leftarrow \mathbf{M}_{\alpha+1} \dots \mathbf{M}_{(\lg N - \ell)}$ 
6:    $\mathbf{M}_B \leftarrow 2^{-(\lg N - \ell)} \mathbf{M}_1 \dots \mathbf{M}_\alpha$ 
7:    $m \leftarrow 2^\alpha$ 
8:   ▷ Part 1: Compute  $\mathbf{a}' = \mathbf{M}_A \hat{\mathbf{a}}$  in  $O(N^2/m) = O(N^2/2^\alpha)$ 
9:    $\mathbf{a}' = \mathbf{0} \in \mathbb{Z}_p^N$ 
10:  for  $j = 0$  to  $m - 1$  do
11:    for  $k = 0$  to  $N/m - 1$  do
12:       $\mathbf{a}'[jm + k] \leftarrow \sum_{i=0}^{N/m-1} \mathbf{M}_A[jN/m + k][i] \cdot \hat{\mathbf{a}}[jN/m + i]$ 
13:  ▷ Part 2: Compute  $\mathbf{a} = \mathbf{M}_B \mathbf{a}'$  in  $O(Nm) = O(N2^\alpha)$ 
14:   $\mathbf{a} = \mathbf{0} \in \mathbb{Z}_p^N$ 
15:  for  $i = 0$  to  $N - 1$  do
16:    for  $k = 0$  to  $m - 1$  do
17:       $j \leftarrow i + kN/m \bmod N$ 
18:       $\mathbf{a}[i] \leftarrow \mathbf{a}[i] + \mathbf{M}_B[i][j] \cdot \mathbf{a}'[j]$ 
19:  return  $\mathbf{a} \in \mathbb{Z}_p^N$ 
20: end procedure

```

**Algorithm 2.** Two-part inverse  $\ell$ -incomplete NTT.

split into more than two parts, the error growth would be too high due to the increased homomorphic multiplication depth. Therefore, we only considered the two-part inverse NTT in our experiments.

## 4.2 Free incompleteness: embedding the base multiplication into Part 1

We know that the incomplete NTT provides some clear advantages compared to the complete formulation: it is faster to compute because of the reduced number of layers and enables us to use a strictly larger set of prime moduli. The only drawback is the increased complexity of the multiplication of polynomials in the incomplete NTT domain, often called the base multiplication, or basemul for short. While it is well-known that, in some lattice-based cryptographic schemes [ABD<sup>+</sup>21, Nat24], the complete NTT can be slower than the incomplete NTT with  $\ell = 1$ , the optimal level of incompleteness can depend on the scheme parameters and hardware characteristics [Kan22]. Interestingly, when using the two-part INTT, we noticed that the base multiplication can be embedded into the first part, i.e., the multiplication by  $\mathbf{M}_A$ . In what follows, we show that it is possible to use the two-part INTT in a way that a small incompleteness level  $\ell$  can be used for fast multiplication without any performance impact.

**Embedding the base multiplication into Part 1.** Suppose we want to compute the product  $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$  of polynomials  $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p[X]/(X^N + 1)$ , and let us assume  $p$  and  $N$

enable the use of  $\ell$ -incomplete NTT. This means the polynomial  $(X^N + 1)$  factors as

$$(X^N + 1) = \prod_{i=1}^{N/2^\ell} (X^{2^\ell} - \psi_i) \pmod{p},$$

for a particular set of elements  $\psi_1, \dots, \psi_{(N/2^\ell)} \in \mathbb{Z}_p$ . In the  $\ell$ -incomplete NTT domain, each polynomial  $\mathbf{a}$  and  $\mathbf{b}$  is represented by a sequence  $\{\hat{\mathbf{a}}_i\}$  and  $\{\hat{\mathbf{b}}_i\}$ , respectively, for  $i = 1$  to  $N/2^\ell$ , where  $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i \in \mathbb{Z}_p[X]/(X^{2^\ell} - \psi_i)$ . In this case, the base multiplications correspond to the computations of the products  $\hat{\mathbf{c}}_i = \hat{\mathbf{a}}_i \cdot \hat{\mathbf{b}}_i$  for all  $i$ . The product polynomial can then be computed using the inverse NTT as  $\mathbf{c} = \text{INTT}(\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \dots, \hat{\mathbf{c}}_{N/2^\ell})$ .

Since we use the two-part inverse NTT based on matrix-vector multiplications in the amortized bootstrapping application, let us put the base multiplications in the same form. The base multiplication is typically done using a naive schoolbook multiplication algorithm as it is often the best choice for small values of  $\ell$  [Kan22]. Notice that the schoolbook multiplication of polynomials  $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i \in \mathbb{Z}_p[X]/(X^{2^\ell} - \psi_i)$  can be easily represented in the matrix-vector form as follows. Let  $\hat{\mathbf{c}}_i = \hat{\mathbf{a}}_i \cdot \hat{\mathbf{b}}_i$  and let  $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i, \hat{\mathbf{c}}_i \in \mathbb{Z}_p^{2^\ell}$  be the vectors of coefficients of polynomials  $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i$ , and  $\hat{\mathbf{c}}_i$ , respectively. Then,  $\hat{\mathbf{c}}_i = \mathbf{A}_i \hat{\mathbf{b}}_i$ , where

$$\mathbf{A}_i = \begin{bmatrix} \hat{\mathbf{a}}_i[0] & \hat{\mathbf{a}}_i[n-1]\psi_i & \cdots & \hat{\mathbf{a}}_i[1]\psi_i \\ \hat{\mathbf{a}}_i[1] & \hat{\mathbf{a}}_i[0] & \cdots & \hat{\mathbf{a}}_i[2]\psi_i \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{a}}_i[n-1] & \hat{\mathbf{a}}_i[n-2] & \cdots & \hat{\mathbf{a}}_i[0] \end{bmatrix} \in \mathbb{Z}_p^{2^\ell \times 2^\ell}.$$

If we choose parameter  $\alpha$  in  $0 \leq \alpha \leq \lg N - \ell$ , we can define the 2-part INTT matrices  $\mathbf{M}_B$  and  $\mathbf{M}_A$  as discussed in the previous section. Furthermore, define the block diagonal matrix  $\mathbf{A}$  as

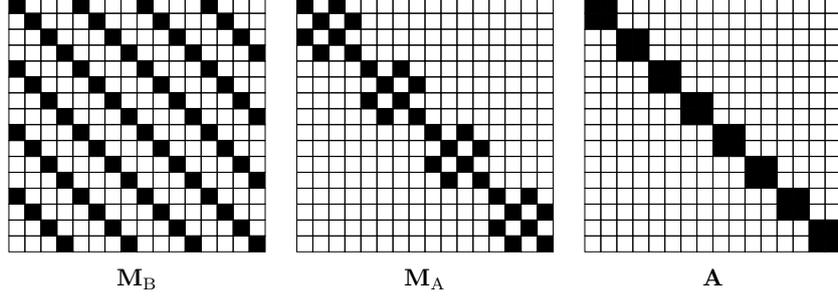
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{A}_{(N/2^\ell)} \end{bmatrix} \in \mathbb{Z}_p^N.$$

Then, if  $\mathbf{c} \in \mathbb{Z}_p^N$  is the vector of coefficients of the product  $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ , we obtain the matrix description of the inverse NTT as  $\mathbf{c} = \mathbf{M}_B \mathbf{M}_A \hat{\mathbf{A}} \hat{\mathbf{b}}$ .

Notice that  $\mathbf{M}_A \mathbf{A}$  is a product of two block diagonal matrices, where the square blocks in  $\mathbf{M}_A$  and  $\mathbf{A}$  have size  $(N/2^\alpha)$  and  $2^\ell$ , respectively, as illustrated in Figure 2. Since  $N$  is a power of 2, both dimensions are also powers of 2, which means  $\mathbf{M}_A \mathbf{A}$  is also a block matrix with square blocks of size  $\max\{N/2^\alpha, 2^\ell\}$ . However, since  $\alpha \leq \lg N - \ell$ , then  $\max\{N/2^\alpha, 2^\ell\} = N/2^\alpha$ . Therefore, if  $\mathbf{X} = \mathbf{M}_A \mathbf{A}$ , the computation of the matrix-vector product  $\mathbf{X}\mathbf{y}$  has exactly the same complexity as computing  $\mathbf{M}_A \mathbf{y}$ , for an arbitrary vector  $\mathbf{y}$ . We then say that matrix  $\mathbf{X} = \mathbf{M}_A \mathbf{A}$  embeds the base multiplication into the first part of the two-part INTT.

**Incompleteness for free.** Suppose we use the complete NTT formulation for a prime  $p$  and a power-of-two  $N$ , and let us assume  $\alpha^*$  is the balancing factor that yields the best performance of the two-part INTT. Let  $p'$  be a prime for which we can use the  $\ell$ -incomplete negacyclic NTT for polynomials in  $\mathbb{Z}_{p'}[X]/(X^N + 1)$ . Then, as long as  $\ell \leq \lg N - \alpha^*$ , if we use the two-part INTT with embedding of the base multiplication to Part 1, the operation count for these two procedures is the same:

1. computing the pointwise product between two vectors in the NTT domain followed by the complete INTT for fast multiplication in  $\mathbb{Z}_p[X]/(X^N + 1)$ ;



**Figure 2.** Shapes of the matrices  $\mathbf{M}_A$ ,  $\mathbf{M}_B$  and  $\mathbf{A}$  for  $N = 16$ ,  $\ell = 1$  and  $\alpha = 2$ . White and black squares represent the positions of null and non-null values, respectively.

2. computing the base multiplication of vectors in the  $\ell$ -incomplete NTT domain followed by the incomplete INTT for fast multiplication in  $\mathbb{Z}_{p'}[X]/(X^N + 1)$ .

This observation is at the core of our contribution, since it enables us to leverage incompleteness to use smaller primes. While on one hand, the number of operations is the same even when using smaller primes – on the other hand, homomorphic operations modulo smaller primes can be processed faster.

### 4.3 Homomorphic polynomial multiplication using the incomplete NTT

We now give a full description of the homomorphic implementation of our multiplication algorithm. To do that, we use the homomorphic dot product presented in [GPVL23, Algorithm 7] as a building block. This function, denoted as `homomorphic_dot_product`, uses the encryption  $\mathbf{K}_{sk}$  of the secret key  $sk$ , which we recall is the secret key used for the  $\tilde{\mathcal{R}}_Q$ GSW encryptions, and the set of key-switching keys  $\{\mathbf{K}_v\}_{v=0}^{p-1}$  together with automorphisms  $X \mapsto X^v$  to compute the dot product between a plaintext vector  $\mathbf{u} \in \mathbb{Z}_p^k$  and an encrypted vector  $\mathbf{m} \in \mathbb{Z}_p^k$ . Concretely, if  $\mathbf{m}, \mathbf{u} \in \mathbb{Z}_p^k$  and each  $\mathbf{C}_i$  encrypts  $X^{m_i}$ ,

$$\mathbf{C}' \leftarrow \text{homomorphic\_dot\_product}\left(\{\mathbf{C}_i\}_{i=1}^k, \mathbf{u}, \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1}\right)$$

results in an encryption  $\mathbf{C}'$  of  $X^{\langle \mathbf{u}, \mathbf{m} \rangle}$ .

Algorithm 3 shows the homomorphic implementation of the polynomial multiplication using the two-part  $\ell$ -incomplete NTT. The algorithm homomorphically multiplies the polynomial  $\mathbf{a} \in \hat{\mathcal{R}}_p$  by an encrypted polynomial  $\mathbf{z}\mathbf{k}$  as follows. More precisely, the core inputs are the polynomial  $\mathbf{a} \in \hat{\mathcal{R}}_p$  and the GSW encryptions  $\mathbf{Z}_i$  of each coefficient  $\hat{\mathbf{z}}_i$  of the NTT representation of the polynomial  $\mathbf{z}\mathbf{k} \in \hat{\mathcal{R}}_p$ . It also receives the encryption  $\mathbf{K}_{sk}$  of the secret key  $sk$ , and a set  $\{\mathbf{K}_v\}$  of key-switching keys, used to perform the dot products. The last parameters are the incompleteness parameter  $\ell$  and the balancing parameter  $\alpha$ , which define the two-part incomplete NTT.

The algorithm begins by performing some non-homomorphic operations, which have negligible cost compared to the homomorphic ones. First, it computes the auxiliary matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$  and the base multiplication embedding matrix  $\mathbf{X}$ . Then it performs two rounds of homomorphic dot products, first for the multiplication by  $\mathbf{X}$ , then for the multiplication by  $\mathbf{M}_B$ . Each element  $\mathbf{C}_i$  from the output  $(\mathbf{C}_0, \dots, \mathbf{C}_{N-1})$  consists of the encryption of  $X^{c_i}$  where  $c_i$  is the  $i$ th coefficient of the polynomial product  $\mathbf{a} \cdot \mathbf{z}$ .

```

1: procedure homomorphic_polymul( $\mathbf{a}, \{\mathbf{Z}_i\}_{i=0}^{N-1}, \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1}, \ell, \alpha$ )
  ▶  $\mathbf{a} \in \tilde{\mathcal{R}}_p$ 
  ▶  $\{\mathbf{Z}_i\}_{i=0}^{N-1}$  are the bootstrapping keys, where each  $\mathbf{Z}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{-z[i]})$ 
  ▶  $\mathbf{K}_{sk} \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(-sk)$ 
  ▶ Each  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_{sk}(sk(X^v))$  is a key-switching key for the automorphism  $X \mapsto X^v$ 
  ▶  $\ell \in \mathbb{Z}$  is the incompleteness level (compatible with  $N$  and  $p$ )
  ▶  $\alpha \in \mathbb{Z}$  is the balancing parameter between parts 1 and 2

2:   ▶ Build matrices  $\mathbf{M}_A$  and  $\mathbf{M}_B$  (can be precomputed for fixed  $(N, p, \ell, \alpha)$ )
3:    $\mathbf{M} \leftarrow$  Matrix associated with the  $\ell$ -incomplete INTT of length  $N$  in  $\mathbb{Z}_p$ 
4:    $\mathbf{M}_1, \dots, \mathbf{M}_{(\lg N - \ell)} \leftarrow$  Butterfly matrices that factor  $\mathbf{M}$ 
5:    $\mathbf{M}_A \leftarrow \mathbf{M}_{\alpha+1} \dots \mathbf{M}_{(\lg N - \ell)}$ 
6:    $\mathbf{M}_B \leftarrow 2^{-(\lg N - \ell)} \mathbf{M}_1 \dots \mathbf{M}_\alpha$ 

7:   ▶ Build matrix  $\mathbf{X}$  embedding the base multiplication by  $\hat{\mathbf{a}}$  into  $\mathbf{M}_A$ 
8:    $\hat{\mathbf{a}} \leftarrow \text{NTT}(\mathbf{a}) \in \mathbb{Z}_p^N$ 
9:    $\mathbf{A} \leftarrow$  block matrix representing the base multiplication by  $\hat{\mathbf{a}}$ 
10:   $\mathbf{X} \leftarrow \mathbf{M}_A \mathbf{A}$ 
11:  ▶ No homomorphic operations used yet, so all code above has negligible impact on
    performance

12:  ▶ Part 1: Homomorphic computation of  $\mathbf{X}\hat{\mathbf{z}}$  in  $O(N^2/2^\alpha)$ 
13:  for  $i = 0$  to  $N - 1$  do
14:     $\mathbf{C}'_i \leftarrow \text{homomorphic\_dot\_product}(\mathbf{Z}_i, \mathbf{X}[i], \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1})$ 
15:  ▶ Part 2: Homomorphic computation of  $\mathbf{M}_B(\mathbf{X}\hat{\mathbf{z}})$  in  $O(N2^\alpha)$ 
16:  for  $i = 0$  to  $N - 1$  do
17:     $\mathbf{C}_i \leftarrow \text{homomorphic\_dot\_product}(\{\mathbf{C}'_i\}_{i=0}^{N-1}, \mathbf{M}_B[i], \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1})$ 
18:  return  $\{\mathbf{C}_i\}_{i=0}^{N-1}$            ▶ Each  $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{(a \cdot zk)[i]})$ 
19: end procedure

```

**Algorithm 3.** Proposed homomorphic polynomial multiplication using the two-part incomplete INTT formulation. This function can be plugged into the partial decryption function defined in the GPVL algorithm [GPVL23, Algorithm 9].

#### 4.4 The bootstrapping algorithm

Finally, we present our amortized bootstrapping algorithm in Algorithm 5, which closely follows the structure of [GPVL23, Algorithm 11], with the only difference occurring in the homomorphic decryption procedure. Indeed, we refer to [GPVL23, Algorithm 16] for `PackLWE` and [GPVL23, Algorithm 10] for `MsgExtract`, respectively, as we mentioned in Section 3 that these algorithms are somewhat standard and not the focus of either our work or [GPVL23]. The last procedure, `NTTDecModified`, is very similar to [GPVL23, Algorithm 9], but the homomorphic polynomial multiplication step is executed with our Algorithm 3. The pseudocode for it is given in Algorithm 4.

We now provide an analysis of the cost, error growth, and correctness of our algorithm, which naturally closely follows the results presented in [GPVL23].

**Theorem 1** (Cost of the bootstrapping algorithm, from [GPVL23] Lemma 9). *Let  $Q = \prod_{i=1}^u q_i$  be the ciphertext modulus and  $d$  be the number of CRT digits used in the GSW ciphertexts. Given at most  $N$  LWE ciphertexts and the relevant bootstrapping keys, Algorithm 5 outputs LWE ciphertexts at the cost of  $O(N^{1+1/\rho} \cdot \rho \cdot d^2 \cdot u)$  NTTs and  $O(N^{1+1/\rho} \cdot \rho \cdot d \cdot u^2 \cdot p)$  multiplications in  $\mathbb{Z}_{q_i}$  where  $\rho = \lg N/\alpha$ .*

```

1: procedure NTTDecModified( $(\mathbf{a}, \mathbf{b}), \{\mathbf{Z}_i\}_{i=0}^{N-1}, \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1}, \ell, \alpha$ )
    ▶  $(\mathbf{a}, \mathbf{b}) \in \tilde{\mathcal{R}}_p^2$  is the packed ciphertext
    ▶  $\{\mathbf{Z}_i\}_{i=0}^{N-1}$  are the bootstrapping keys, where each  $\mathbf{Z}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{-z^{[i]}})$ 
    ▶  $\mathbf{K}_{sk} \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(-sk)$ 
    ▶ Each  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_{sk}(sk(X^v))$  is a key-switching key for the automorphism  $X \mapsto X^v$ 
    ▶  $\ell \in \mathbb{Z}$  is the incompleteness level (compatible with  $N$  and  $p$ )
    ▶  $\alpha \in \mathbb{Z}$  is the balancing parameter between parts 1 and 2
2:    $\{\mathbf{C}_i\}_{i=0}^{N-1} \leftarrow \text{homomorphic\_polymul}(\mathbf{a}, \{\mathbf{Z}_i\}_{i=0}^{N-1}, \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1}, \ell, \alpha)$ 
3:   for  $i = 0$  to  $N - 1$  do
4:      $\tilde{\mathbf{C}}_i \leftarrow X^{b^{[i]}} \cdot \mathbf{C}_i$ 
5:   return  $(\tilde{\mathbf{C}}_0, \dots, \tilde{\mathbf{C}}_{N-1})$            ▶ Each  $\tilde{\mathbf{C}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{(a \cdot zk + b)^{[i]}})$ 
6: end procedure

```

**Algorithm 4.** Homomorphic partial decryption.

*Proof.* In [GPVL23], the radix  $m$  is set to  $m = N^{1/\rho}$  where  $\rho$  is the recursive depth of the bootstrapping algorithm. In our work, we set  $m = 2^\alpha$  and thus  $\rho = \lg N/\alpha$ . The overall cost of Algorithm 5 is asymptotically dominated by the cost of NTTDecModified and within this algorithm, the cost of homomorphic\_polymul. Both Algorithm 1 (the original two-part inverse NTT computation) and Algorithm 3 (homomorphic\_polymul) have the same asymptotic cost where Part 1 uses  $O(N^2/2^\alpha)$  homomorphic operations and Part 2 uses  $O(N2^\alpha)$  homomorphic operations. Thus NTTDecModified has the same complexity as NTTDec, and the overall complexity of the algorithm remains unchanged from [GPVL23].  $\square$

Noise growth is another important factor of bootstrapping as it directly influences the correctness of the algorithm. The only difference between Algorithm 5 and [GPVL23, Algorithm 11] is the NTTDec procedure, more precisely, the homomorphic polynomial multiplication within it.

**Theorem 2** (Noise growth of the bootstrapping algorithm). *Given at most  $N$  LWE ciphertexts and the relevant bootstrapping keys, Algorithm 5 outputs refreshed LWE ciphertexts with at most the same noise as given in [GPVL23, Lemma 9].*

*Proof.* Given  $N$  input LWE ciphertexts with associated initial error, the resulting error of the packed ring ciphertext after applying PackLWE remains the same as in [GPVL23] as no modification is done to this initial step of the amortized bootstrapping algorithm (we refer to [GPVL23, Algorithm 16] for the exact expression of the associated error). To prove that we expect a similar final error, it then suffices to show that the noise associated with the GSW ciphertexts  $\mathbf{C}_i$  outputted by NTTDec is the same as the noise of the equivalent GSW ciphertexts outputted by NTTDecModified. The final outputted noise after MsgExtract follows from [GPVL23, Lemma 11].

In Algorithm 4, NTTDecModified, the noise growth solely comes from the call to homomorphic\_polymul as the multiplication by  $X^{b^{[i]}}$  modulo  $X^p - 1$  (line 4) will only rotate the coefficients of the noise term.

We also know that both Algorithm 1 (the original two-part inverse NTT computation) and Algorithm 3 (homomorphic\_polymul) follow a very similar structure and have the same number of homomorphic operations along with the same inputs. Thus it is reasonable to assume that the noise growth resulting from Algorithm 1 and Algorithm 3 will be the same as given by [GPVL23, Lemma 7].

Finally, we note that NTTDec needs to account for noise coming from a key switching operation (line 6) after applying the GSW Galois automorphisms (line 5). Our algorithm

```

1: procedure Bootstrap $\left(\left(\mathbf{c}_0, \dots, \mathbf{c}_N\right), \left(f_0, \dots, f_N\right), \left\{\mathbf{Z}_i\right\}_{i=0}^{N-1}, \mathbf{K}_{sk}, \left\{\mathbf{K}_v\right\}_{v=0}^{p-1}, \ell, \alpha\right)$ 
    ▶ Each  $\mathbf{c}_i \in \text{LWE}_s^{p^*}(\Delta \cdot m_i) \in \mathbb{Z}_p^N$  for  $0 \leq i < N$  and  $\Delta = \lfloor p^*/2^\kappa \rfloor$ 
    ▶ Each  $f_i : \mathbb{Z}_{2^\kappa} \rightarrow \mathbb{Z}_{2^\kappa}$  is the function that will be applied to the  $i$ th packed ciphertext
    ▶  $\{\mathbf{Z}_i\}_{i=0}^{N-1}$  are the bootstrapping keys, where each  $\mathbf{Z}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{-\mathbf{z}^{[i]}})$ 
    ▶  $\mathbf{K}_{sk} \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(-sk)$ 
    ▶ Each  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_{sk}(sk(X^v))$  is a key-switching key for the automorphism  $X \mapsto X^v$ 
    ▶  $\ell \in \mathbb{Z}$  is the incompleteness level (compatible with  $N$  and  $p$ )
    ▶  $\alpha \in \mathbb{Z}$  is the balancing parameter between parts 1 and 2
2:    $(\mathbf{a}, \mathbf{b}) \leftarrow \text{PackLWE}(\mathbf{c}_0, \dots, \mathbf{c}_{N-1})$ 
3:    $(\mathbf{a}', \mathbf{b}') \leftarrow \text{ModSwitch}_{Q \rightarrow p}(\mathbf{a}, \mathbf{b})$ 
4:    $(\mathbf{C}_0, \dots, \mathbf{C}_{N-1}) = \text{NTTDecModified}\left((\mathbf{a}', \mathbf{b}'), \{\mathbf{Z}_i\}_{i=0}^{N-1}, \mathbf{K}_{sk}, \{\mathbf{K}_v\}_{v=0}^{p-1}, \ell, \alpha\right)$ 
5:   ▷ Each  $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{sk}(X^{e_i + \Delta \cdot m_i})$  for small error  $e_i$ 
6:   for  $i = 0$  to  $N - 1$  do
7:      $\mathbf{c}'_i \leftarrow \text{MsgExtract}(\mathbf{C}_i, f_i)$ 
8:   return  $(\mathbf{c}'_0, \dots, \mathbf{c}'_{N-1})$ 
9: end procedure

```

**Algorithm 5.** Amortized bootstrapping algorithm for message space  $\mathbb{Z}_{2^\kappa}$ .

NTTDecModified does not require this operation and thus does not need to account for the error growth resulting from it.<sup>4</sup> □

**Corollary 1** (From [GPVL23] Corollary 2). *The bootstrapping algorithm presented in Algorithm 5 has noise overhead of  $\tilde{O}(\lambda^{1.5+\rho})$ , where  $\lambda$  is the security parameter and  $\rho = \lg N/\alpha$ .*

*Proof.* This follows from Theorem 2 and arguments given in [GPVL23, Corollary 2]. □

**Theorem 3** (Correctness of bootstrapping, adapted from [GPVL23] Theorem 1). *Let  $\lambda$  be a security parameter and  $p^* = \tilde{O}(\lambda^{2.5+\rho})$  for  $\rho = \lg N/\alpha$ . Consider the input ciphertexts  $\mathbf{c}_i \in \text{LWE}_s^{p^*}(\Delta \cdot m_i, E) \in \mathbb{Z}^{p^*}$  encrypting messages  $m_i \in \mathbb{Z}_{2^\kappa}$  for  $0 \leq i < N$  with initial error satisfying  $E = O(p^*/\lambda)$  and  $\Delta = \lfloor p^*/2^\kappa \rfloor$ . Then, with probability  $1 - 2^{-\lambda}$ , the output of Algorithm 5 is correct, i.e., it outputs valid LWE encryptions of  $f(m_i)$  for arbitrary functions  $f : \mathbb{Z}_{2^\kappa} \rightarrow \mathbb{Z}_{2^\kappa}$ .*

*Proof.* We refer to the proof of [GPVL23, Theorem 1] for the technical details. □

## 5 Experimental results

In this section, we present the experimental results of our work. We first recall the advantages of the use of incompleteness for NTT and in particular illustrate the wide range of moduli that can be chosen as a function of the incompleteness degree  $\ell$ . We then present our best timings and trade-offs between execution time and DFR and compare our results with the results given in [GPVL23] and with the state-of-the-art sequential algorithm TFHE-rs.

<sup>4</sup>Note that, in the original code, these extra key switching operations are avoided by multiplying the twiddle factors directly by the corresponding coefficients of  $\hat{\mathbf{a}} = \text{NTT}(\mathbf{a})$ .

## 5.1 How incompleteness enables faster bootstrapping and lower DFR

We now discuss how incompleteness can help us achieve more efficient or reliable bootstrapping by reducing the constraints on  $p$ . Unfortunately, these goals oppose each other: there are parameters that achieve very low decryption failure rates but whose performance is impractical, and vice-versa.

Let us assume that, by the requirements of some particular application, both the number  $N$  of LWE ciphertexts that are packed, and the bit-length  $\kappa$  of the plaintext messages are fixed. In this case, the most critical parameter affecting the bootstrapping performance is the prime  $p$ , since most operations are done in the circulant ring  $\tilde{\mathcal{R}} = \mathbb{Z}[X]/(X^p - 1)$ . On the one hand, for performance, smaller values of  $p$  are better, as the reduced number of coefficients will entail less costly operations. On the other hand, interestingly, regarding the DFR, the most critical parameter is also  $p$ , as can be seen in Equation 1 from Section 3.2.<sup>5</sup> However in this case, to reduce the DFR, we want  $p$  as large as possible.

Recall that to accelerate operations in  $\tilde{\mathcal{R}}$  using HEXL optimized implementations, Guimarães et al. [GPVL23] use the auxiliary ring  $\tilde{\mathcal{R}}' = \mathbb{Z}[X]/(X^{N'} - 1)$ , where  $N' = 2^{\lceil \lg p \rceil + 1}$ , as described in Section 3.3. Therefore, to significantly improve the bootstrapping performance, it is not enough to simply reduce  $p$ . Indeed, we have to reduce it to prime values with a smaller number of bits. This has an important consequence: when considering DFR and performance together, the optimal choices of  $p$  are close to, but below, powers of two. This ensures we are taking somewhat large values of  $p$  (to guarantee a reasonable DFR), without crossing over to the next power of two (which would cause a significant performance impact).

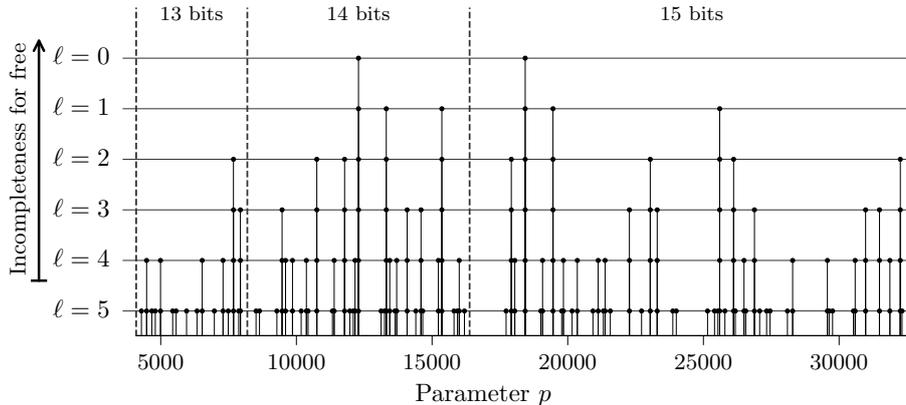
The advantage provided by the incomplete NTT for our amortized bootstrapping is that it allows us to select values that are close to powers of two. Figure 3 shows how the increased incompleteness levels  $\ell$  allows us to find parameters much closer to powers of two than the complete NTT ( $\ell = 0$ ), considering the negacyclic NTT over  $\mathbb{Z}_p[X]/(X^N + 1)$  for  $N = 1024$ . Our proposed technique allows us to keep the performance similar for all primes between the powers of two, represented by dotted vertical bars in Figure 3. So, without impacting the performance, from the pool of primes with a fixed number of bits, we have the flexibility to choose an optimal prime that is the closest to the bars to achieve the best DFR. For concreteness, for the range of interest between 13-bit and 15-bit primes, there are only two possible values of  $p$  when considering a complete NTT: 12289 (14-bit), which is what is used by Guimarães et al. [GPVL23] and 18433 (15-bit). However, using the incomplete NTT with  $\ell \leq 4$  we can find significantly better parameters, such as 16001 and 32257, for the 14-bit and 15-bit ranges, respectively.

In the next section, we explore the possible tradeoffs between performance and DFR. In particular, we consider the different values  $p$  possible by the incomplete NTT and their interaction with other parameters such as  $N$ ,  $p^*$  or the Hamming weight of the secret key in order to achieve significantly better results than what Guimarães et al. [GPVL23] obtain for the complete NTT.

## 5.2 Comparison with Guimarães et al.

We tested our algorithm with different sets of parameters varying the ring  $\hat{\mathcal{R}}$  size  $N$ , the prime moduli  $p$  and  $p^*$ , and the hamming weight  $w$  of the secret key. We run all our experiments on the same machine using an Intel Xeon Platinum 8260 CPU at 2.4 GHz with 250 GB of RAM, running Ubuntu 20.04.6 LTS, and report results for single-threaded executions. All the parameter sets considered achieve at least a 128-bit security level based on the LWE estimator [APS15]. We report our results in Table 2, where the amortized

<sup>5</sup>We remark that  $p^*$  and  $w$  that appear in the definition of  $\varepsilon$  used in Equation 1 also impact the DFR, but their impact is somewhat limited since the square root of  $\varepsilon$  is considered.



**Figure 3.** Possible prime values of  $p \equiv 1 \pmod{2N/2^\ell}$  that enable the use of  $\ell$ -incomplete NTT for fast multiplication in  $\mathbb{Z}_p[X]/(X^N + 1)$  for  $N = 1024$  and different incompleteness levels  $\ell$ . For  $N = 1024$ , the optimal balance parameter is  $\alpha^* = 6$ , which means that incompleteness levels  $\ell \leq \lg N - \alpha^* = 4$  come for free in this case.

execution time (in milliseconds) is the median observed over 10 experiments (with standard deviation at most 1.3%). The DFR for  $\kappa$ -bit messages is computed using Equation 1. The first row of Table 2 shows the parameters and results obtained using [GPVL23]’s code with no modifications. The table shows the smallest values of  $\ell$  for which prime  $p$  is compatible with  $N$ , i.e.,  $p \equiv 1 \pmod{2N/2^\ell}$ .

With the flexibility and benefits of incompleteness, we are able to lower the modulus  $p$  to 7937 and obtain a much improved amortized time of 584 ms for  $N = 1024$ . This results in a  $2.12\times$  speedup compared to [GPVL23]. However, in this case, our DFR is worse than in the original work and as high as  $2^{-9}$  for 8-bit messages. When considering 7-bit messages, the DFR lowers to  $2^{-31.7}$ . In order to improve upon the DFR for 8-bit messages while still maintaining good timings, we increase the ring size to  $N = 2048$ . In this case, we are able to decrease the DFR significantly to  $2^{-39.1}$  while still improving on the amortized time for  $p = 7681$ . More precisely, we obtain an amortized time of 879ms, corresponding to a  $1.41\times$  speedup compared to [GPVL23].

Overall, Table 2 showcases an interesting tradeoff between the execution time of our algorithm and the observed DFRs. We identify the best values in bold. Indeed, as we are able to explore a wide range of modulus values  $p$ , we can significantly lower the execution time of the algorithm by considering much smaller primes, however, at the cost of a high DFR, but also significantly lower the DFR by increasing the prime modulus  $p$ . For both  $N = 1024$  and  $N = 2048$ , by choosing  $p$  larger than the original modulus 12289, we are able to improve the DFR of [GPVL23] by  $2^{-7}$  and  $2^{-41}$ , respectively, with only a small effect on performance. More precisely, by choosing  $p = 16001$  and  $p = 15361$  for  $N = 1024$  and  $N = 2048$  respectively, we obtain DFR values of  $2^{-24}$  and  $2^{-131}$  (while the original algorithm obtains  $2^{-17}$  for 8-bit messages) while the performance drops of  $0.94\times$  and  $0.64\times$  respectively. In the case of  $N = 2048$  and  $p = 15361$ , we even obtain a better DFR than the  $2^{-128}$  failure probability guaranteed by TFHE-rs v1.0 (see also [BJSW24] for technical details). We also note that when considering 9-bit messages, we achieve a DFR of at most  $2^{-35}$  by considering the largest prime  $p = 15361$  and  $N = 2048$ .

**Table 2.** Parameter sets offering different tradeoffs between performance and decryption failure rate (DFR). All parameters considered satisfy a 128-bit security level. The execution time is shown for single-threaded runs. While the parameter set for  $N = 2048$  and  $\ell = 0$  was not proposed by GPVL, it can be used with their original code. All parameters use  $\alpha = 6$  (the optimal value for the complete NTT), which means every incompleteness level  $\ell \leq 4$  comes for free for  $N = 1024$  and 2048.

Algorithm	$N$	$p$	$\lg(p^*)$	$w$	$\ell$	DFR for $\kappa$ -bit messages			Amortized time (ms)	Speedup
						$\kappa = 7$	$\kappa = 8$	$\kappa = 9$		
[GPVL23]	1024	12289	24	256	0	$2^{-62.4}$	$2^{-17.3}$	$2^{-5.4}$	1,237.9	–
Ours	1024	7681	24	256	2	$2^{-30.0}$	$2^{-8.8}$	$2^{-3.0}$	585.6	$2.11\times$
	1024	7937	24	256	3	$2^{-31.7}$	$2^{-9.3}$	$2^{-3.1}$	<b>584.7</b>	<b><math>2.12\times</math></b>
	1024	16001	24	256	4	$2^{-88.8}$	$2^{-24.1}$	$2^{-7.2}$	1,311.1	$0.94\times$
	2048	7681	27	52	3	$2^{-147.8}$	<b><math>2^{-39.1}</math></b>	$2^{-11.2}$	<b>879.8</b>	<b><math>1.41\times</math></b>
	2048	7937	27	52	4	$2^{-157.2}$	$2^{-41.5}$	$2^{-11.8}$	882.7	$1.40\times$
	2048	12289	27	52	0	$2^{-350.3}$	$2^{-90.2}$	$2^{-24.4}$	1,872.5	$0.66\times$
	2048	15361	27	52	2	$2^{-516.3}$	$2^{-131.9}$	<b><math>2^{-35.0}</math></b>	1,931.5	$0.64\times$

**Table 3.** Comparison with state-of-the-art implementations of sequential and amortized bootstrapping algorithms. For TFHE-rs, we run `pbs128-bench`. All experiments are single-threaded.

Bootstrapping type	Algorithm	Source	LWE secret key type	Time per LWE ciphertext (ms)	Number of ciphertexts per bootstrapping
Sequential	TFHE PBS	TFHE-rs	Binary	660.68	1
	[GPVL23]	Code	Ternary	1,255.5	1024
Amortized	Ours	Code	Ternary	584.7	1024
	Ours	Code	Ternary	879.8	2048

### 5.3 Comparison with state-of-the-art

In addition to the improvement over [GPVL23] already reported in Table 2, we compare our algorithm with its sequential counterpart for programmable bootstrapping, namely TFHE-rs [Zam22], the latest version v1.0 (from February 2025). In Table 3, we report the amortized cost in milliseconds. Similarly to before, the experiments were all run on the same machine with single-threaded executions. For our algorithm, we provide two timings: our fastest time using  $N = 1024$  but where the DFR is high ( $2^{-9}$  as reported in Table 2 for 8-bit messages) and our fastest time for  $N = 2048$  where the increased ring size allows us to lower the DFR significantly ( $2^{-39}$  as reported in Table 2 for 8-bit messages) while still achieving better time than [GPVL23]. For the sequential algorithm benchmarks, the only implementation for programmable (or functional) bootstrapping is TFHE-rs with a running time of 660ms, which is only  $1.33\times$  faster than our timing with reasonable DFR, i.e.,  $2^{-39}$ . Our best time is actually  $1.12\times$  faster than TFHE-rs but this is ignoring the high DFR for 8-bit messages. If we consider 7-bit messages, the speedup remains  $1.12\times$  faster than TFHE-rs and in this case, our DFR is about  $2^{-32}$ . Note that when considering 7-bit messages, our algorithm is  $1.12\times$  faster than TFHE-rs while the DFR has the acceptable value of  $2^{-31.7}$ . We leave for future work the exploration of techniques to lower the DFR while maintaining our best timings.

## 6 Conclusion and future work

In this work, we present an improved amortized bootstrapping algorithm for FHEW/TFHE ciphertexts, building on the algorithm introduced by Guimarães et al. [GPVL23]. While the asymptotic complexity of our algorithm remains the same as in [GPVL23] and [DMKMS24], we introduced algorithmic refinements that yield practical speedups. In particular, we leveraged the notion of incompleteness in the NTT to expand the set of usable moduli and explore trade-offs between amortized runtime and decryption failure rates (DFRs). Our alternative two-part incomplete NTT formulation embeds the base multiplication into the first part, allowing the use of incomplete NTT with no additional performance cost. Although we observe significant speedups compared to [GPVL23], as well as to the sequential TFHE-rs implementation, our best timings come at the cost of higher DFRs. Reducing the DFR while maintaining comparable performance remains an open question and presents a promising direction for future research.

Additionally, we note that [GPVL23] relies on the HEXL library for optimization, which supports only power-of-two cyclotomic rings. Future improvements to amortized bootstrapping may benefit from extending support to arbitrary cyclotomics in widely used FHE libraries such as OpenFHE. Our work, aligned with recent advancements, further highlights the potential of amortized bootstrapping algorithms to outperform sequential counterparts or, at the very least, serve as viable alternatives in advancing the practicality of FHE.

## Acknowledgements

We thank the authors of [GPVL23] for addressing our questions regarding their work, and in particular, the first author, Antonio Guimarães, for clarifying specific details in both the paper and the accompanying code.

## References

- [ABC19] Erdem Alkım, Yusuf Bilgin, and Murat Cenk. Compact and simple rlwe based key encapsulation mechanism. In *Progress in Cryptology–LATINCRYPT 2019*, pages 237–256. Springer, 2019.
- [ABD<sup>+</sup>19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.
- [ABD<sup>+</sup>21] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02), 2021. <https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf>.
- [APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
- [BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In *Progress in Cryptology–AFRICACRYPT 2018: 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10*, pages 217–251. Springer, 2018.

- [BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
- [BJSW24] Olivier Bernard, Marc Joye, Nigel P Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. *Cryptology ePrint Archive*, 2024.
- [BKS<sup>+</sup>21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, and Vinodh Gopal. Intel HEXL: Accelerating homomorphic encryption with Intel AVX512-IFMA52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 57–62, 2021.
- [Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual cryptology conference*, pages 868–886. Springer, 2012.
- [CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.
- [CHK<sup>+</sup>21] Chi Chung, Vincent Hwang, Matthias Kannwischer, Gregor Seiler, Cheng Shih, and Bo Yang. NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 159–188, 2021.
- [CP05] Richard Crandall and Carl Pomerance. *Prime numbers: a computational perspective*, volume 2. Springer, 2005.
- [DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.
- [DMKMS24] Gabrielle De Micheli, Duhyeong Kim, Daniele Micciancio, and Adam Suhl. Faster amortized FHEW bootstrapping using ring automorphisms. In *IACR International Conference on Public-Key Cryptography*, pages 322–353. Springer, 2024.
- [FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.
- [GP25] Antonio Guimarães and Hilder V. L. Pereira. Fast amortized bootstrapping with small keys and polynomial noise overhead. *Cryptology ePrint Archive*, Paper 2025/686, 2025.
- [GPVL23] Antonio Guimarães, Hilder VL Pereira, and Barry Van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–35. Springer, 2023.
- [GS66] William Morven Gentleman and Gordon Sande. Fast Fourier Transforms: For fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578, 1966.

- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.
- [Kan22] Matthias J. Kannwischer. *Polynomial multiplication for post-quantum cryptography*. PhD thesis, Radboud University, 2022. <https://kannwischer.eu/thesis/phd-thesis-2022-05-09.pdf>.
- [Lip21] BAM Lips. The efficiency of polynomial multiplication methods for ring-based PQC algorithms of round 3 of the NIST PQC competition. Master’s thesis, Eindhoven Univ. of Technology, 2021.
- [LMK<sup>+</sup>23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 227–256. Springer, 2023.
- [LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 1–23. Springer, 2010.
- [LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, May 2019.
- [LW23a] Feng-Hao Liu and Han Wang. Batch bootstrapping I: A new framework for SIMD bootstrapping in polynomial modulus. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 321–352. Springer, 2023.
- [LW23b] Feng-Hao Liu and Han Wang. Batch bootstrapping II: Bootstrapping in polynomial modulus only requires  $\tilde{O}(1)$  FHE multiplications in amortization. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 353–384. Springer, 2023.
- [LW23c] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with  $\tilde{O}(1)$  polynomial multiplications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 101–132. Springer, 2023.
- [Moe76] Robert Moenck. Practical fast polynomial multiplication. In *Proc. of the 3rd ACM symposium on Symbolic and algebraic computation*, pages 136–148. ACM, 1976.
- [MS18] Daniele Micciancio and Jessica Sorrell. Ring Packing and Amortized FHEW Bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:14. Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

- [Nat24] National Institute of Standards and Technology. FIPS203: Module-lattice-based key-encapsulation mechanism standard. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Aug 2024. <https://doi.org/10.6028/NIST.FIPS.203>.
- [Nus80] Henri Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, 1980.
- [SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 617–635. Springer, 2009.
- [Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. <https://github.com/zama-ai/tfhe-rs>.