




# Amortized Bootstrapping Revisited: Simpler, Asymptotically-faster, Implemented

Antonio Guimarães<sup>1</sup>, Hilder V. L. Pereira<sup>2</sup>, and Barry van Leeuwen<sup>2</sup>

<sup>1</sup> University of Campinas, Brazil,

<sup>2</sup> imec-COSIC, KU Leuven, Leuven, Belgium.

antonio.guimaraes@ic.unicamp.br,  
hildervitor.limapereira@kuleuven.be,  
barry.vanleeuwen@kuleuven.be

**Abstract.** Micciancio and Sorrel (ICALP 2018) proposed a bootstrapping algorithm that can refresh many messages at once with sublinearly many homomorphic operations per message. However, despite the attractive asymptotic cost, it is unclear if their algorithm can be practical, which reduces the impact of their results. In this work, we follow their general framework, but propose an amortized bootstrapping that is conceptually simpler and asymptotically cheaper. We reduce the number homomorphic operations per refreshed message from  $O(3^\rho \cdot n^{1/\rho} \cdot \log n)$  to  $O(\rho \cdot n^{1/\rho})$ , and the noise overhead from  $\tilde{O}(n^{2+3\cdot\rho})$  to  $\tilde{O}(n^{1.5+\rho})$ . To obtain a concrete instantiation of our bootstrapping algorithm, we propose a double-CRT (aka RNS) version of the GSW scheme, including a new operation, called *shrinking*, used to speed-up homomorphic operations by reducing the dimension and ciphertext modulus of the ciphertexts. We provide a C++ implementation of our algorithm, thus showing that the amortized bootstrapping is not only theoretical, but practical. Moreover, it is up to 2.7 times faster than an equivalent non-amortized version for the smallest parameter set we consider, and gains are expected to increase as the parameters increase.

## 1 Introduction

Since the introduction of the first Fully Homomorphic Encryption (FHE) scheme, by Gentry [Gen09], there has been a quest to improve the efficiency and the security of FHE. The main efficiency bottleneck of any FHE scheme is the *bootstrapping*, which is an operation that refreshes the ciphertexts after they have been involved in a few homomorphic operations, allowing us thus to perform further operations on them. Hence, most works aiming to make FHE more efficient directed the efforts towards designing faster bootstrapping algorithms. For this goal, there are two main strategies: Heavy packed bootstrapping and fast single-message bootstrapping.

In the first strategy, one tries to pack several messages into a “large” ciphertext, then the bootstrapping becomes complex and very costly, but refreshes several messages at once, thus, the *amortized* cost per message is expected to be not very high. This type of bootstrapping was proposed for many schemes [BGV12,FV12,BGH13,GHS12], obtaining good amortized costs, however also offering the counter-point that these schemes generally were not very efficient regarding the noise management, thus, their bootstrapping algorithms often incur in quasi-polynomial noise growth, which implies that their security is based on worst-case lattice problems with *superpolynomial approximation factors*. Ideally, we would like to have FHE with assumptions as weak as the ones used for general lattice-based public-key encryption, namely, worst-case lattice problems with *polynomial approximation factors* only.

On the other front, one encrypts a single message into a “small” ciphertext, thus, hugely simplifying the bootstrapping, making it possible to execute it much faster, in many cases, in a few

Scheme	Total cost	Messages	Amortized cost	Noise overhead
[DM15]	$\tilde{O}(n)$	1	$\tilde{O}(n)$	$\tilde{O}(n^{1.5})$
[CGGI16]	$O(n)$	1	$O(n)$	$\tilde{O}(n^1)$
[MS18]	$\tilde{O}(3^\rho \cdot n^{1+1/\rho})$	$O(n)$	$\tilde{O}(3^\rho \cdot n^{1/\rho})$	$\tilde{O}(n^{2+3\cdot\rho})$
This work	$O(\rho \cdot n^{1+1/\rho})$	$O(n)$	$O(\rho \cdot n^{1/\rho})$	$\tilde{O}(n^{1.5+\rho})$

**Table 1:** Comparison of number of homomorphic operations and noise growth of bootstrapping algorithms of different schemes based on worst-case lattice problems with polynomial approximation factor. The notation  $\tilde{O}$  hides polylogarithmic factors in  $n$ .

milliseconds on a common commercial computer [DM15,CGGI16,Per21,BIP<sup>+</sup>22]. The downside now is the need of one bootstrapping per gate of the circuit being evaluated homomorphically and, since each bootstrapping refreshes a single message, the amortized cost is still somehow high. One advantage of this bootstrapping strategy is that it introduces only a polynomial noise overhead, allowing thus to reduce the security of those schemes to worst-case lattice problems with polynomial approximation factors.

Then, in [MS18], Micciancio and Sorrell try to obtain the advantages of these two approaches, by proposing a bootstrapping algorithm that follows the blueprint of [DM15], but packs several messages into a single ciphertext to amortize the cost of the bootstrapping. Therewith, they obtain the first FHE scheme whose security is based on the hardness of worst-case lattice problems with polynomial approximation factors that at the same time is bootstrappable with amortized sublinearly many homomorphic operations. Their main idea is to describe the bootstrapping as a polynomial multiplication, then to evaluate it homomorphically using some fast polynomial multiplication algorithm. For technical reasons, due to the limitations of the functions we can evaluate homomorphically, they cannot simply evaluate a Fast Fourier Transform, thus, they adapt the Nussbaumer Transform [Nus80] to work over power-of-three cyclotomic rings, then use it in their algorithm to bootstrap  $O(n)$  messages in time  $\tilde{O}(3^\rho \cdot n^{1+1/\rho})$ , where  $\rho$  is a free parameter, and the notation  $\tilde{O}$  hides polylogarithmic factors on  $n$ . Therefore, their amortized cost is only  $\tilde{O}(3^\rho \cdot n^{1/\rho})$  homomorphic operations per message.

Following the blueprint of [MS18], we propose a simpler and more efficient amortized bootstrapping. Our first contribution is to remove the Nussbaumer Transform, replacing it by a standard (homomorphic) Number Theoretic Transform (NTT). By doing so, we not only make the whole bootstrapping algorithm more straightforward, but we also gain important asymptotic factors, decreasing the number of homomorphic operations per message from  $O(3^\rho \cdot n^{1/\rho} \cdot \log n)$  to  $O(\rho \cdot n^{1/\rho})$ , and the noise introduced by the bootstrapping from  $\tilde{O}(n^{2+3\rho})$  to  $\tilde{O}(n^{1.5+\rho})$ . In Table 1, we present a comparison of our work with previous ones.

Although [MS18] obtains a significant asymptotic improvement over previous works, it is unclear how (in)efficient it would be in practice, since the hidden constants are hard to estimate. Thus, as a second contribution, we present a concrete instantiation of our method. For this, we propose a double-CRT (RNS) variant of the GSW scheme [GSW13], including a new operation, called *shrinking*, that allows to efficiently reduce the ciphertext size, and thus, the cost of the homomorphic operations, as the noise grows. We stress that this contribution is also of independent interest, as many primitives or protocols that use the GSW scheme can benefit from an RNS version of it. This also allows us to present a concrete cost analysis, in terms of polynomial multiplications (or NTTs),

which gives us a much better idea of the practical cost of the amortized bootstrapping and makes it easier to compare with other works, since the number times that the NTT is executed is already used to estimate the cost of several previous schemes, such as [DM15,CGGI16,BIP<sup>+</sup>22].

Finally, we also implemented our bootstrapping in C++ and made it publicly available. Thus, as a third contribution, we present baseline running times and memory usage for this type of amortized bootstrapping and we show that it is actually doable in practice, with running times comparable to some existing schemes.

Now we show more technical details about the amortized bootstrapping, our techniques, and our results.

## 1.1 Overview of the amortized bootstrapping from [MS18]

The bootstrapping strategy of [AP14], improved and made practical in [DM15], works as follows: the whole FHE scheme is organized in two layers, each one composed by one homomorphic scheme. We have an LWE-based scheme that can perform very limited number of homomorphic operations, then has to be bootstrapped. We call it the base scheme. Then we have a second scheme, called the accumulator, that is used to evaluate the decryption of the base scheme homomorphically, i.e., to bootstrap it. For the accumulator, one uses the GSW scheme [GSW13] instantiated with the RLWE problem, so that it can encrypt polynomials. Because of the slow noise growth of GSW, the noise overhead of the bootstrapping just a polynomial. Essentially, to decrypt an LWE ciphertext  $\mathbf{c}$ , one has to multiply it by the secret key  $\mathbf{s}$ . Thus, starting with GSW encryptions of powers of  $X$  with the secret key in the exponent, i.e.,  $X^{s_i}$ , the GSW homomorphic multiplications are used to compute  $\prod_{i=0}^n X^{c_i \cdot s_i} = X^{\mathbf{c} \cdot \mathbf{s}}$ . Finally, there is an extraction procedure that maps this power of  $X$  to the message encrypted by  $\mathbf{c}$ . Notice that the bootstrapping costs  $\tilde{O}(n)$  homomorphic operations, more specifically, GSW multiplications.

The main idea of [MS18] is to combine  $O(n)$  LWE ciphertexts into one single RLWE ciphertext  $\mathbf{c} \in \mathcal{R}^2$  encrypting  $O(n)$  messages. Then, because in the RLWE problem the secret is a polynomial  $s$  instead of a vector, decrypting  $\mathbf{c}$  now boils down to performing a polynomial multiplication on  $\mathcal{R}$ , which can be done in time  $O(n \cdot \log n)$  via standard techniques, such as the Fast Fourier Transform (FFT). Thus, if we could use the accumulator to evaluate an FFT, the amortized cost of such bootstrapping would be only  $O(\log n)$  homomorphic operations per message. However, due to limitations in the noise growth of this bootstrapping strategy, it is not possible to evaluate all the  $O(\log n)$  recursive levels of the FFT. Thus, [MS18] sets the recursion level as a parameter  $\rho$ .

Moreover, since the GSW scheme is instantiated over the ring  $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1 \rangle$  and we only work with powers of  $X$ , whose order is  $2N$  in  $\mathcal{R}$ , there is a limited set linear operations over  $\mathbb{Z}_{2N}$  available as homomorphic operations. So, for example, we cannot take an encryption of  $(X$  to the power of)  $m$  and produce an encryption of  $-m$  or of  $m^{-1}$ . Therefore, [MS18] cannot evaluate an FFT. To overcome this limitation, they pack the LWE ciphertexts into an RLWE ciphertext defined over a power-of-three cyclotomic ring, i.e., defined modulo  $\Phi_{3^k}(X) = 2 \cdot 3^{k-1} + 3^{k-1} + 1$ , and adapt the Nussbaumer transform to replace the FFT and perform polynomial multiplications modulo  $\Phi_{3^k}(X)$ . The radix  $r$  Nussbaumer transform works as the FFT, by dividing the input by  $r$  in each recursive level. However, in their adapted algorithm, there is an expansion by 3, i.e., instead of obtaining  $r$  inputs of length  $(n/r)$ , they obtain  $3r$  inputs. Since this expansion happens in all recursive levels, at the end, their bootstrapping costs  $\tilde{O}(3^\rho \cdot n^{1+1/\rho})$  homomorphic operations and the noise introduced by the bootstrapping is  $\tilde{O}(n^{2+3 \cdot \rho})$ .

## 1.2 Overview of our contributions and techniques

**Simpler and more efficient amortized bootstrapping:** Micciancio and Sorrell accepted that the accumulator constructed with GSW just provides a limited set of operations over  $\mathbb{Z}_{2N}$ , where  $N$  is a power of two, and tried to adapt the fast polynomial multiplication algorithms to work with that instruction set. We diverge from this by trying to adapt the accumulator to the algorithm we want to evaluate, instead of trying to adapt the algorithm to the accumulator. That is, we know that the Number Theoretic Transform (NTT) is the algorithm choice to perform multiplications modulo  $X^N + 1$ , the typically used in the RLWE problem, thus, our goal is to obtain an accumulator that can evaluate NTTs.

To obtain that, we use the results from [BDF18] to instantiate the GSW scheme modulo  $X^p - 1$ , where  $p$  is a prime number, but with security based on the RLWE problem. This gives us essentially the same instruction set of [MS18], but over  $\mathbb{Z}_p$ . Then, we set  $p \equiv 1 \pmod{2N}$ , so that we have a  $2N$ -root of unity in  $\mathbb{Z}_p$  and the NTT of dimension  $N$  is well-defined. Then, we extend recent results about using automorphisms on bootstrapping algorithms [BDF18, LMK<sup>+</sup>22] to the GSW scheme, which expands the instruction set of our accumulator. Putting it all together, we obtain an GSW-based accumulator that allows us to homomorphically evaluate a standard NTT. The only limitation we have then is that the noise overhead of the bootstrapping is still exponential in the number of recursive levels of the NTT, thus, as in [MS18], we only have  $\rho$  recursive levels, so that the noise overhead is still polynomial in  $N$ .

With a more powerful accumulator, the bootstrapping algorithm becomes much simpler, as its main step is essentially the same as a well-known NTT. Moreover, there is no longer the expansion by 3 within the recursions, which allows us to save a factor of  $3^\rho$  in the time complexity and to reduce the noise overhead from  $\tilde{O}(n^{2+3\cdot\rho})$  to  $\tilde{O}(n^{2+\rho})$ .

Additionally, our accumulator also allows us to replace the algorithm used in [MS18] to perform the entry-wise vector multiplication in the FFT domain, called SlowMult, by a cheaper and simpler procedure, which yields an additional gain of a  $\log n$  factor. Therefore, we reduce the number of homomorphic operations from  $O(3^\rho \cdot n^{1+1/\rho} \cdot \log n)$  in [MS18] to  $O(\rho \cdot n^{1+1/\rho})$ .

**Double-CRT version of GSW:** FHE schemes implementing single-message bootstrapping, such as [DM15, CGGI16, BIP<sup>+</sup>22], can use very small parameters when compared to other FHE schemes, thanks to the almost linear noise overhead of the bootstrapping. In particular, the ciphertext modulus,  $Q$ , is typically an integer smaller than  $2^{32}$ . Since all homomorphic operations are defined modulo  $Q$ , these schemes can be implemented using 32-bit integers, which are native types of most CPUs.

For other schemes, the ciphertext modulus,  $Q$ , is much larger, normally with more than one thousand bits. Thus, implementing the operations modulo  $Q$  requires more care: one represents  $Q$  as a product of small primes  $q_i$ 's, typically with less than 32 bits, then uses the Chinese Remainder Theorem (CRT) to express operations modulo  $Q$  as independent operations modulo each  $q_i$ , allowing thus to use again the native integer types. Moreover, all the polynomials composing the ciphertexts are also stored in the FFT domain. Since the FFT can be seen as a type of CRT, this representation is often called double-CRT [HS15]. An alternative name for this is the RNS representation [HPS19a].

Since the amortized bootstrapping has at least quadratic noise overhead, it also typically requires  $Q$  with more bits than native types of CPUs. Therefore, to obtain a practical implementation, we propose a double-CRT version of the accumulator, i.e., the GSW scheme. This includes all common

operations already existing for GSW, such as homomorphic multiplication and external product, and also new operations, like Galois automorphisms and key switchings.

One optimization that is commonly used for GSW, is to ignore least significant bits of the ciphertexts during the multiplications, as they correspond to the noise of the RLWE samples. For example, in the TFHE scheme [CGGI16], this is called approximate decomposition and it is used to speed up the bootstrapping. However, on the double-CRT representation, there is no notion of least significant bits, and this technique no longer applies. Thus, we propose a *ciphertext shrinking*, which introduces the implementation of approximate gadget decompositions over the double-CRT representation. It takes a GSW ciphertext, which is a  $2d \times 2$  matrix where each entry is a polynomial modulo  $Q$ , and outputs another GSW ciphertext as a  $2d' \times 2$  matrix and defined modulo  $Q'$ , where  $d' < d$  and  $Q' < Q$ , with basically the same (relative) noise. Reducing simultaneously  $d$  and  $Q$  enables a cubic performance improvement in all core homomorphic operations.

The shrinking operation has the side effect of multiplying the message by some integer scaling factor. Thus, we have to generalize the definition of the GSW ciphertexts to keep track of these scaling factors, which can be removed during the homomorphic evaluation without any impact in the noise or on the efficiency of the scheme.

We notice that any protocol or scheme that uses GSW can benefit from our double-CRT representation. Thus, this contribution is of independent interest. For example, both versions of [LMK<sup>+</sup>22] could exploit RNS representation to obtain bootstrapping for large messages.

Thanks to this lower-level description of the GSW scheme, we can also estimate the cost of our amortized bootstrapping in a more concrete way, namely, in terms of NTTs and integer (modular) multiplications. This simplifies the comparison with other bootstrapping strategies and also clarifies the practicability of the amortized bootstrapping.

**Further optimizations and proof-of-concept implementation in C++:** We provide the first implementation of a bootstrapping algorithm for FHE based on the worst-case hardness of lattice problems with polynomial approximation factors with amortized sublinearly many homomorphic operations. We show that our construction is practical, being up to 2.7 times faster than the non-packed approach in the smallest parameter set we tested. Our implementation also shows that the shrinking technique enables up to 1.7 times speedup in the homomorphic INTT evaluation for slightly larger parameters.

For the implementation, we propose several further optimization techniques to our general description of the amortized bootstrapping. For example, as the LWE-to-RLWE packing becomes a fundamental part of the bootstrapping procedure, we introduce a new strategy for minimizing its noise overhead. Essentially, we evaluate the dimension reduction key switching, which commonly precedes all bootstrapping methods, in two parts, with the packing key switching between them. In this way, we can produce the input RLWE sample with noise almost as low as we would have for a non-packed bootstrap.

Our source code is publicly available, since we believe that this can help the academic community to understand our techniques and also simplify comparisons in future works. We stress that the description of [MS18] is very high level and also that any implementation of their bootstrapping, even using our double-CRT GSW scheme, must be far from practical, due to all the hidden constants in the asymptotic costs. As an example, it is not clear how much memory they would need, since it is hard to estimate practical parameters for their scheme. Thus, one could reasonably wonder if

the amortized bootstrapping would ever be practically feasible, and our algorithms together with our implementation provides a positive answer.

## 2 Preliminaries

For  $a_1, \dots, a_k, m_1, \dots, m_k \in \mathbb{Z}$ , such that all the  $m_i$ 's are coprime, let  $M = \prod_{i=1}^k m_i$  and define  $\text{CRT}_{m_1, \dots, m_k}(a_1, \dots, a_k)$  as the unique  $a \in \mathbb{Z}_M$  such that  $a_i = a \bmod m_i$ . Also, for any  $a \in \mathbb{Z}_M$ , define  $\text{CRT}_{m_1, \dots, m_k}^{-1}(a) = (a \bmod m_1, \dots, a \bmod m_k)$ . For an element  $a(X)$  of any polynomial ring of the form  $\mathbb{Z}[X]/\langle f(X) \rangle$ , we extend CRT and  $\text{CRT}^{-1}$  by applying it coefficient wise.

For any vector  $\mathbf{u}$ , we denote the infinity norm by  $\|\mathbf{u}\|$  and the Euclidean norm by  $\|\mathbf{u}\|_2$ . For any polynomial  $a = \sum_{i=0}^d a_i X^i$ , we define the norm of  $a$  as the norm of the coefficient vector  $(a_0, \dots, a_d)$ . If  $a$  is an element of a polynomial ring like  $\mathbb{Z}[X]/\langle f(X) \rangle$ , we consider  $a' \in \mathbb{Z}[X]$  as the unique canonical representation of  $a$ , and thus the norm of  $a$  is simply the norm of  $a'$ .

### 2.1 Rings

We use power-of-two cyclotomic rings of the form  $\mathbb{Z}[X]/\langle X^N + 1 \rangle$ , where  $N = 2^k$  for some  $k \in \mathbb{N}$ , which we denote by  $\hat{\mathcal{R}}$ , and circulant rings of the form  $\mathbb{Z}[X]/\langle X^p - 1 \rangle$ , for some prime number  $p$ , which we denote by  $\tilde{\mathcal{R}}$ . For any positive integer  $Q$ , we define  $\hat{\mathcal{R}}_Q := \hat{\mathcal{R}}/(Q\hat{\mathcal{R}})$  and  $\tilde{\mathcal{R}}_Q := \tilde{\mathcal{R}}/(Q\tilde{\mathcal{R}})$ , i.e., the same rings as before, but with coefficients of the elements reduced modulo  $Q$ .

### 2.2 Plain, ring and circulant LWE

In the well-known learning with errors problem (LWE) [Reg05] with parameters  $n, q$ , and  $\sigma$ , an attacker has to find a secret vector  $\mathbf{s} \in \mathbb{Z}^n$  given many samples of the form  $(\mathbf{a}_i, b_i)$ , where  $\mathbf{a}_i$  is uniformly sampled from  $\mathbb{Z}_q^n$  and  $b_i := \mathbf{a}_i \cdot \mathbf{s} + e_i \bmod q$ , with  $e_i$  following a discrete Gaussian distribution with parameter  $\sigma$ .

The ring version of LWE, known as RLWE [LPR10], is used to obtain more efficient cryptographic schemes, since it typically allows us to encrypt larger messages when compared to similar schemes instantiated with LWE. In the RLWE, we fix the ring  $\mathcal{R} = \mathbb{Z}[X]/\langle \Phi_m(X) \rangle$ , where  $\Phi_m(X)$  is the  $m$ -th cyclotomic ring, and we are given samples of the form  $(a_i, b_i)$ , where  $a_i$  is uniformly sampled from  $\mathcal{R}_q$  and  $b_i := a_i \cdot s + e_i \bmod q$ , where the  $e_i$  is a small noise term, and we have to find the secret polynomial  $s$ . Most schemes are constructed on top of the RLWE problem with a power-of-two cyclotomic polynomial,  $\Phi_{2N}(X) = X^N + 1$ , where  $N = 2^k$  for some  $k \in \mathbb{N}^*$ .

In this work, we also use a variant of the LWE called circulant-LWE (CLWE), which was introduced in [BDF18] and was proved to be as hard as the RLWE on prime-order cyclotomic polynomials. Hence, we restrict ourselves to prime  $p$ . Instead of using the ring  $\mathcal{R} = \mathbb{Z}[X]/\langle \Phi_p(X) \rangle$ , we use the ‘‘circulant ring’’  $\tilde{\mathcal{R}} = \mathbb{Z}[X]/\langle X^p - 1 \rangle$ . Then, CLWE samples are obtained essentially by projecting RLWE samples from  $\mathcal{R}$  to  $\tilde{\mathcal{R}}$ . This is done by fixing some integer  $Q$  prime with  $p$  and by defining the map  $L_Q : \mathcal{R}_Q \rightarrow \tilde{\mathcal{R}}_Q$  as

$$L_Q : \sum_{i=0}^{p-1} a_i \cdot X^i \mapsto \sum_{i=0}^{p-1} a_i \cdot X^i - p^{-1} \cdot \left( \sum_{i=0}^{p-1} a_i \right) \cdot \sum_{i=0}^{p-1} X^i \bmod Q$$

Finally, given an RLWE sample  $(a', b' = a' \cdot s' + e') \in \mathcal{R}_Q^2$ , we define the corresponding CLWE sample as  $(a, b) := (L_Q(a'), L_Q((1 - X) \cdot b')) \in \tilde{\mathcal{R}}_Q^2$ . Thanks to the homomorphic properties of  $L_Q$ ,



we have  $b = a \cdot s + e \bmod Q$ , where  $e = L_Q((1 - X) \cdot e')$  is a small noise term and  $s = L_Q((1 - X) \cdot s')$  is the CLWE secret.

Then, using the CLWE problem, the GSW instantiated over the circulant ring  $\tilde{\mathcal{R}}$  is CPA-secure if we restrict the message space to powers of  $X$ , that is, if one just encrypts  $X^k \in \tilde{\mathcal{R}}$  for  $k \in \mathbb{Z}$  [BDF18].

In Section 3.5, we extend the results [BDF18] so that we can also encrypt non-powers of  $X$ , as this is needed in our bootstrapping algorithm, specially, to use Galois automorphisms on GSW ciphertexts.

## 2.3 Subgaussian distributions

A random variable  $X$  is subgaussian with parameter  $\sigma > 0$ , in short,  $X$  is  $\sigma$ -subgaussian, if for all  $t \in \mathbb{R}$  it holds that  $\mathbb{E}[\exp(2\pi tX)] \leq \exp(\pi\sigma^2 t^2)$ . The name subgaussian comes from the fact that the “tails” of  $X$  are bounded by a Gaussian function of standard deviation  $\sigma$ , that is,  $X$  is  $\sigma$ -subgaussian  $\implies \forall t \in \mathbb{R}, \Pr[|X| \geq t] \leq 2\exp(-\pi t^2/\sigma^2)$ . This allows one to bound the absolute value of  $X$  with overwhelming probability. Namely, by setting  $t = \sigma\sqrt{\lambda/\pi}$ , we see that  $\Pr[|X| \geq \sigma\sqrt{\lambda/\pi}] \leq 2\exp(-\pi(s\sqrt{\lambda/\pi})^2/s^2) = 2\exp(-\lambda) < 2^{-\lambda}$ . In other words, the Gaussian tails tell us that  $|X|$  is smaller than  $\sigma\sqrt{\lambda/\pi}$  with probability at least  $1 - 2^{-\lambda}$ , which is exponentially close to 1. If  $X$  is a  $B$ -bounded centered distribution, then  $X$  is  $(\sigma \cdot \sqrt{2\pi})$ -subgaussian. Linear combinations of independently distributed subgaussians are again subgaussians, i.e., given independent  $\sigma_i$ -subgaussian distributions  $X_i$ ’s, then for any  $\mathbf{c} = (c_1, \dots, c_n) \in \mathbb{R}^n$ , it holds that  $Y := \sum_{i=1}^n c_i X_i$  is  $(\sqrt{\sum_{i=1}^n c_i^2 \sigma_i^2})$ -subgaussian. In particular, if all  $\sigma_i$  are equal to some  $\sigma$ , then,  $Y$  is  $(\sigma \cdot \|\mathbf{c}\|_2)$ -subgaussian. Also, since  $\|\mathbf{c}\|_2 \leq \sqrt{n} \cdot \|\mathbf{c}\|_\infty$ , we have  $Y$  is  $(\sigma \cdot \sqrt{n} \cdot \|\mathbf{c}\|_\infty)$ -subgaussian. We say that a polynomial  $a$  is  $\sigma$ -subgaussian if its coefficients are independent  $\sigma_i$ -subgaussian, with  $\sigma_i \leq \sigma$ . For any  $n \in \mathbb{N}^*$ , given  $f$  equal to  $X^n \pm 1$  and two polynomials  $a$  and  $b$  of degree less than  $n$  following subgaussians with parameters  $\sigma_a$  and  $\sigma_b$ , we can see that for  $c = a \cdot b \bmod f$ , we have  $c_i$  is  $(\sqrt{n} \cdot \sigma_a \cdot \sigma_b)$ -subgaussian. However, the coefficients  $c_i$ ’s are not necessarily independent, thus, to say that  $c$  is  $(\sqrt{n} \cdot \sigma_a \cdot \sigma_b)$ -subgaussian, we use the independence heuristic.

## 2.4 Independence heuristic

To derive average bounds for the noise growth of the ciphertexts, we will use the well-known and commonly used independence heuristic: All the coefficients of the noise terms of the LWE and RLWE samples appearing in the linear combinations we consider are assumed to be independent and concentrated. In more detail, they follow  $\sigma$ -subgaussian, where  $\sigma^2$  is their variance.

## 2.5 RNS (double-CRT) representation for polynomial arithmetic

Homomorphic operations of commonly used FHE schemes are composed by some operations over polynomial rings  $\mathcal{R}_Q = \mathbb{Z}_Q[X]/\langle f(X) \rangle$ , where  $f(X)$  is a degree- $N$  polynomial over  $\mathbb{Z}_Q[X]$ . Here, we suppose that  $f(X) = X^N + 1$  or  $f(X) = X^N - 1$ . These operations essentially boil down to adding and multiplying polynomials of degree less than  $N$ , then reducing them modulo  $f(X)$ , and finally reducing each coefficient modulo  $Q$ .

Because  $Q$  generally has much more than 64 bits, working with elements of  $\mathcal{R}_Q$  requires libraries that implement arbitrary precision integers, which is inefficient. To overcome this, the residual

number system (RNS), aka double-CRT, is typically used. It exploits the decomposition of  $Q$  to work with several polynomials modulo each  $q_i$ , which then fit in the 32- or 64-bit native integer types of current processors.

In more detail, because  $Q = \prod_{i=1}^{\ell} q_i$ , by using the Chinese remainder theorem coefficient-wise, we have

$$\mathcal{R}_Q = \mathbb{Z}_Q[X]/\langle f(X) \rangle = \prod_{i=1}^{\ell} \mathbb{Z}_{q_i}[X]/\langle f(X) \rangle$$

Thus, additions and multiplications over  $\mathcal{R}_Q$  can be implemented with  $\ell$  independent operations over  $\mathcal{R}_{q_i}$ . Moreover, since multiplying polynomials modulo  $f(X)$  requires fast Fourier transforms or number-theoretic transforms (NTT), it is common to go one step forward and represent elements of  $\mathcal{R}_Q$  in the “NTT form”, i.e., given  $a(X) \in \mathcal{R}_Q$ , instead of simply storing its list of coefficients, we define  $a_i(X) := a(X) \bmod q_i$  and precompute  $\mathbf{a}_i = \text{NTT}_{q_i}(a(X))$ , for every  $q_i$ . Notice that we need to choose  $q_i$  such that a suitable primitive root of unity  $\omega_i \in \mathbb{Z}_{q_i}$  exists. Then, we have the following is an isomorphism:

$$\text{NTT}_{q_i}(a(X)) = (a(\omega_i^0), a(\omega_i^1), \dots, a(\omega_i^{N-1})) \in \mathbb{Z}_{q_i}^N$$

So, given  $a(X) \in \mathcal{R}_Q$ , we store the matrix

$$\text{Mat}(a) := \begin{pmatrix} \text{NTT}_{q_1}(a \bmod q_1) \\ \vdots \\ \text{NTT}_{q_{\ell}}(a \bmod q_{\ell}) \end{pmatrix} = \begin{pmatrix} a_{1,0} & \dots & a_{1,N-1} \\ \vdots & \ddots & \vdots \\ a_{\ell,0} & \dots & a_{\ell,N-1} \end{pmatrix} \in \mathbb{Z}^{\ell \times N}$$

With this representation, each addition and multiplication over  $\mathcal{R}_Q$  is implemented with point-wise operations of the corresponding matrices. For example,  $a \cdot b \in \mathcal{R}_Q$  is

$$\text{Mat}(a) \odot \text{Mat}(b) = \begin{pmatrix} a_{1,0} \cdot b_{1,0} & \dots & a_{1,N-1} \cdot b_{1,N-1} \\ \vdots & \ddots & \vdots \\ a_{\ell,0} \cdot b_{\ell,0} & \dots & a_{\ell,N-1} \cdot b_{\ell,N-1} \end{pmatrix} \in \mathbb{Z}^{\ell \times N}$$

where the operations in the  $i$ -th row are performed modulo  $q_i$ .

Finally, sometimes we also want to switch the modulus  $Q$  to a smaller modulus  $Q'$ . We assume that  $Q'$  divides  $Q$ , so we can write  $Q' = \prod_{i=1}^k q_{j_i}$  for some  $k$ . Then, we want to map an element  $a \in \mathcal{R}_Q$  to some  $a' \in \mathcal{R}_{Q'}$ , which is done by considering  $a$  as an element of  $\mathcal{R}$ , then reducing it modulo  $Q'$ . In the double-CRT representation, this is accomplished by simply ignoring the rows of  $\text{Mat}(a)$  corresponding to the prime factors of  $Q/Q'$ . So, if  $Q'$  has  $k$  prime factors, we remove  $\ell - k$  rows of  $\text{Mat}(a)$  and obtain  $\text{Mat}(a') \in \mathbb{Z}^{k \times N}$ .

**Cost of operations in double-CRT representation** We estimate the cost of our algorithms by the number of NTTs and multiplications performed modulo the small primes  $q_i$ 's. We assume that all those primes have about the same bit length, thus, operations modulo any of them cost essentially the same. Moreover, we assume that a forward and a backward NTT modulo  $q_i$  have the same cost, thus, we do not distinguish them in our cost estimations.



## 2.6 Gadget matrix

Consider three positive integers  $Q$ ,  $B$ , and  $d$  such that  $d \in O(\log Q)$ . Let  $\mathbf{g}$  be a  $d$ -dimensional column vector and  $\mathbf{I}_2$  be the  $2 \times 2$  identity matrix. We say that  $\mathbf{G} = \mathbf{I}_2 \otimes \mathbf{g} \in \mathbb{Z}^{2d \times 2}$  is a gadget matrix for the base  $\mathbf{g}$  with quality  $B$  if there is an efficient decomposition algorithm  $G^{-1} : \mathbb{Z}_Q^2 \rightarrow \mathbb{Z}^{2 \times 2d}$  such that, if  $\mathbf{A} := G^{-1}(a, b)$ , then  $\|\mathbf{A}\|_\infty \leq B$  and  $\mathbf{A} \cdot \mathbf{G} = (a, b) \pmod{Q}$ . We naturally extend  $G^{-1}$  to a polynomial ring of the form  $\mathcal{R}_Q = \mathbb{Z}_Q[X]/\langle f(X) \rangle$  by applying  $G^{-1}$  coefficientwise. That is, given  $(a, b) \in \mathcal{R}_Q^2$ , we define  $G^{-1}(a, b) = \sum_{i=0}^{\deg f-1} G^{-1}(a_i, b_i) \cdot X^i$ . We also extend  $G^{-1}$  to matrices  $\mathbf{C} \in \mathcal{R}_Q^{2d \times 2}$  by applying it to each row. Thus, for  $\mathbf{C} \in \mathcal{R}_Q^{2d \times 2}$ , we have  $G^{-1}(\mathbf{C}) \in \mathcal{R}_Q^{2d \times 2d}$ .

The main example of gadget matrix is the one defined by some base  $B \in \mathbb{Z}$ , which corresponds to  $d = \lceil \log_B(Q) \rceil$ ,  $\mathbf{g} = (B^0, B^1, \dots, B^{d-1})^T$ , quality  $B$  and

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ B & 0 \\ \vdots & 0 \\ B^{d-1} & 0 \\ 0 & 1 \\ 0 & B \\ 0 & \vdots \\ 0 & B^{d-1} \end{bmatrix}.$$

In this case, the decomposition  $G^{-1}$  corresponds to the integer decomposition in base  $B$ . For instance, if  $B = 2$ , then  $G^{-1}(a, b) = (a_0, \dots, a_{d-1}, b_0, \dots, b_{d-1})$ , where  $a_i$ 's and  $b_i$ 's are the bits of  $a$  and  $b$ , respectively.

Another example, which will be of central importance in our double-CRT GSW scheme presented in Section 3, is the gadget matrix with respect to a CRT base.

Let  $q_1, \dots, q_\ell$  be prime numbers and let  $Q := \prod_{i=1}^\ell q_i$ . Let  $d \in \mathbb{N}$  be the “number of digits”. For simplicity, assume that  $d \mid \ell$  and let  $k := \ell/d \in \mathbb{Z}$ . Then, for  $1 \leq i \leq d$ , define the  $i$ -th “CRT digit” as  $D_i := \prod_{j=(i-1) \cdot k + 1}^{i \cdot k} q_j$ , that is, a product of  $k$  consecutive primes. Finally, define  $Q_i := Q/D_i$  and  $\hat{Q}_i := (Q/D_i)^{-1} \pmod{D_i}$ . The gadget matrix is now defined as

$$\mathbf{G} = \begin{bmatrix} Q_1 \cdot \hat{Q}_1 & 0 \\ \vdots & 0 \\ Q_d \cdot \hat{Q}_d & 0 \\ 0 & Q_1 \cdot \hat{Q}_1 \\ \vdots & \vdots \\ 0 & Q_d \cdot \hat{Q}_d \end{bmatrix} \in \mathbb{Z}^{2d \times 2}.$$

Then, we define  $G^{-1}(a, b) := (\text{CRT}_{D_1, \dots, D_d}^{-1}(a), \text{CRT}_{D_1, \dots, D_d}^{-1}(b))$ . It is easy to verify that  $G^{-1}(a, b) \cdot \mathbf{G} = (a, b) \pmod{Q}$ . Moreover, since each entry of  $G^{-1}(a, b)$  is of the form  $a \pmod{D_i}$  or  $b \pmod{D_i}$ , we see that the quality of this gadget matrix is  $D := \max(D_1, \dots, D_d)$ .

Because of the ciphertext shrinking that we present in Section 3, we need a more general definition of gadget matrices, which includes an integer scaling factor. Namely, we say that  $\mathbf{G}_\alpha$  is a *scaled gadget matrix* with factor  $\alpha$  if  $G^{-1}(\alpha^{-1} \cdot a, \alpha^{-1} \cdot b) \cdot \mathbf{G}_\alpha = (a, b)$ , in other words, we have to multiply the input  $(a, b)$  by the inverse of  $\alpha$  before decomposing it. In the case of the CRT decomposition, instead of simply using  $Q_i \cdot \hat{Q}_i$ , the entries of  $\mathbf{G}_\alpha$  use  $Q_i \cdot \hat{Q}_i \cdot \alpha_i$  where  $\alpha_i := \alpha \pmod{D_i}$ .

---

**Algorithm 1:** FastBaseExtension

---

**Input:**  $D = \prod_{i=1}^w d_i$ ,  $P = \prod_{i=1}^v p_i$ ,  $a \in \mathcal{R}_D$  in double-CRT form.  
**Output:**  $a' = a + u \cdot D \in \mathcal{R}_{PD}$  in double-CRT form, where  $\|u\| \leq 1/2$ .  
**Complexity:**  $v + w$  NTTs and  $O(v \cdot w \cdot N)$  modular multiplications  
*// Assume that  $\hat{D}_j := (D/d_j)^{-1} \bmod d_j$  are precomputed*

```

1 for  $1 \leq j \leq w$  do
2   Let  $\mathbf{a}^{(j)} := \text{row}_j(\text{Mat}(a)) \in \mathbb{Z}^N$ 
3    $a^{(j)} := \text{NTT}_{d_j}^{-1}(\mathbf{a}^{(j)}) \in \mathcal{R}_{d_j}$ 
4 for  $1 \leq i \leq v$  do
5    $a^{(w+i)} := 0 \in \mathcal{R}_{p_i}$ 
6   for  $1 \leq j \leq w$  do
7      $\text{tmp} := a^{(j)} \cdot \hat{D}_j \bmod d_j$ 
8      $\text{tmp} := \text{tmp} \cdot (D/d_j) \bmod p_i$ 
9      $a^{(w+i)} = (a^{(w+i)} + \text{tmp}) \bmod p_i$ 
10 for  $1 \leq i \leq v$  do
11    $\mathbf{a}^{(w+i)} := \text{NTT}_{p_i}(a^{(w+i)}) \in \mathbb{Z}_{p_i}^N$ 
12 Return  $(\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(w+v)}) \in \mathbb{Z}^{(w+v) \times N}$ .
```

---

## 2.7 Base extension

In some situations, we may want to add or multiply polynomials defined modulo different values  $Q$  and  $D$ . But because all the operations are performed entrywise when we use RNS representation, we first need to represent both operands on a common modulus. This is done with an operation called base extension.

For simplicity, let's assume that  $D = \prod_{i=1}^w d_i$  divides  $Q$  so that we have  $Q = P \cdot D$  for some  $P = \prod_{i=1}^v p_i$ . Then, given  $a \in \mathcal{R}_Q$  and  $b \in \mathcal{R}_D$ , both in double-CRT form, we want to lift  $b$  to  $\mathcal{R}_Q$ . This is done by reconstructing each coefficient of  $b$  modulo  $D$ , then reducing them modulo each  $p_i$ . However, to avoid arbitrary precision integers, we try to reconstruct  $b_i \in \mathbb{Z}_D$  already performing all the operations modulo the  $p_i$ 's. This means that the conversion is not exact and we obtain  $[b_i]_D + u_i \cdot D$  in base  $P$  with  $|u_i| \leq 1/2$  instead of exactly  $[b_i]_D$ . So overall we have the residues of  $[b(X)]_D + u(X) \cdot D$  in the basis  $P \cdot D$ , with  $\|u(X)\|_\infty \leq 1/2$  [KPZ21].

We show this operation in detail in Algorithm 1, but it is essentially defined as follows:

$$\text{FastBaseExtension}(b, D, P) := \left( \sum_{j=1}^w [b \cdot (D/d_j)^{-1}]_{d_j} \cdot (D/d_j) \bmod p_i \right)_{i=1}^v$$

## 2.8 Basic encryption schemes based on LWE, RLWE, and CLWE

We define the set of LWE encryptions of a message  $m \in \mathbb{Z}_t$ , where  $t \geq 2$ , under a secret key  $\mathbf{s} \in \mathbb{Z}^n$ , with  $E$ -subgaussian noise and scaling factor  $\Delta \in \mathbb{Z}$  as

$$\text{LWE}_s^Q(\Delta \cdot m, E) := \{(\mathbf{a}, b) \in \mathbb{Z}_Q^{n+1} : b = [\mathbf{a} \cdot \mathbf{s} + e + \Delta \cdot m]_Q \text{ for some } E\text{-subgaussian } e \in \tilde{\mathcal{R}}\}.$$

For a power-of-two cyclotomic polynomial  $\hat{\mathcal{R}}$ , the set of RLWE ciphertexts encrypting a message  $m \in \hat{\mathcal{R}}$ , with scaling factor  $\Delta \in \mathbb{N}$ , under a secret key  $s$ , and with  $E$ -subgaussian noise is

$$\hat{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m, E) := \{(a, b) \in \hat{\mathcal{R}}_Q^2 : b = [a \cdot s + e + \Delta \cdot m]_Q \text{ for some } E\text{-subgaussian } e \in \hat{\mathcal{R}}\}.$$

Basically the same definition applies to CLWE ciphertexts:

$$\tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m, E) := \{(a, b) \in \tilde{\mathcal{R}}_Q^2 : b = [a \cdot s + e + \Delta \cdot m]_Q \text{ for some } E\text{-subgaussian } e \in \tilde{\mathcal{R}}\}.$$

In any of the three types of ciphertexts, the decryption is done by multiplying the term  $a$  (or  $\mathbf{a}$ ) by the secret key and subtracting it from  $b$  modulo  $Q$ , which gives us  $e' = e + \Delta \cdot m \bmod Q$ , then we output  $\lfloor e'/\Delta \rfloor \bmod t$ . If  $\|e - \Delta \cdot m\| < Q/2$ , then the decryption correctly outputs  $m \bmod t$ .

**Modulus switching** Here, we just consider the case where we want to switch the ciphertext modulus  $Q = \prod_{i=0}^{\ell-1} q_i$  to a smaller  $Q'$  by removing some primes, say,  $q_0, \dots, q_{k-1}$ . Thus,  $Q' = \prod_{i=k}^{\ell-1} q_i$ . It is possible to switch to a larger modulus  $Q'$  by adding new primes to the moduli chain, but since we do not use this in our work, we do not discuss it here.

Let  $q := Q/Q' = q_0 \dots q_{k-1}$ . Given  $\mathbf{c} := (a, b := a \cdot s + e + \Delta m) \in \text{RLWE}_{s,Q}(m)$  where  $\Delta = \lfloor Q/t \rfloor$  for some  $t$ , the modulus switching from  $Q$  to  $Q'$  consists in computing

$$\mathbf{c}' := \lfloor \lfloor \mathbf{c} \cdot Q'/Q \rfloor \rfloor_{Q'} = (\lfloor \lfloor a/q \rfloor \rfloor_{Q'}, \lfloor \lfloor b/q \rfloor \rfloor_{Q'}).$$

Generally,  $\mathbf{c}'$  is an RLWE encryption of the same message  $m$ , but with ciphertext modulus  $Q'$  and noise close to  $e/q$ .

---

**Algorithm 2:** ModSwitch Algorithm

---

**Input:**  $\mathbf{c} = (a, b) \in \text{RLWE}_{s,Q}(m)$ ,  $Q'$  such that  $Q'|Q$

**Output:**  $\mathbf{c}' \in \text{RLWE}_{s,Q'}(m)$

**Complexity:**  $2\ell$  NTT,  $O(k\ell N)$  multiplications on  $\mathbb{Z}_{q_i}$

**Noise growth:**  $E \rightarrow O(E/q + \sqrt{N} \cdot S)$

- 1  $\delta_a = \text{FastBaseExtension}_{Q',q}(a) \bmod q$
  - 2  $\delta_b = \text{FastBaseExtension}_{Q',q}(b) \bmod q$
  - 3  $\hat{\mathbf{c}} = (a - \delta_a, b - \delta_b)$
  - 4  $\mathbf{c}' = \lfloor \hat{\mathbf{c}}/q \rfloor_{Q'} = ((a - \delta_a) \cdot q^{-1}, (b - \delta_b) \cdot q^{-1}) \bmod Q'$
- 

Because rounding is not compatible with double-CRT representation, we avoid it by first subtracting  $\lfloor \mathbf{c} \rfloor_q$  from  $\mathbf{c}$ , so that the result is a multiple of  $q$ , then we can divide by  $q$  and no rounding is needed. In more detail, we define  $\delta_a := a \bmod q$  and  $\delta_b := b \bmod q$ , then compute  $\hat{\mathbf{c}} = (a - \delta_a, b - \delta_b)$ . Notice that all the coefficients of  $\hat{\mathbf{c}}$  belong to  $q\mathbb{Z}$ . Finally, we output  $\mathbf{c}' := \lfloor \hat{\mathbf{c}}/q \rfloor_{Q'} = ((a - \delta_a) \cdot q^{-1}, (b - \delta_b) \cdot q^{-1}) \bmod Q'$ .

To subtract  $\delta_a$  from  $a$ , we first need to use **FastBaseExtension** from  $q$  to  $Q'$  to obtain  $\delta_a$  in base  $Q$ . The same is needed for  $b$ . Each **FastBaseExtension** costs  $\ell$  NTTs and  $O(k\ell N)$  modular multiplications, where  $N$  is the degree of the modulus polynomial. Multiplying by  $q^{-1}$  modulo  $Q'$  means that we have to multiply each residue of  $\mathbf{c}'$  modulo  $q_i$ , for  $k \leq i \leq \ell - 1$  by the inverse of  $q$  modulo  $q_i$ . This step just costs  $O(\ell - k)$  integer modular multiplications. Thus, the overall number of operations  $2\ell$  NTTs and  $O(k\ell N)$  multiplications on  $\mathbb{Z}_{q_i}$ .

After modulus switching, there is a  $\epsilon$  such that  $\|\epsilon\|_\infty \leq 1$  and the noise changes from  $e$  to

$$e' = (e + \delta_a \cdot s - \delta_b)/q + m \cdot \epsilon.$$

Because both  $a$  and  $b$  are (indistinguishable from) uniform modulo  $Q$ , both  $\delta_a$  and  $\delta_b$  are uniform modulo  $q$ , thus, they are  $(q \cdot \sqrt{2\pi})$ -subgaussians. Therefore, if  $e$  is  $(E)$ -subgaussian and

$s$  is  $(S)$ -subgaussian, we have  $e'$  is  $(E/q + \sqrt{N} \cdot S \cdot \sqrt{2\pi} + 2\sqrt{2\pi})$ -subgaussian, where  $N$  is the degree of the modulus polynomial. Simplifying, the modulus switching changes the noise from an  $E$ -subgaussian to a  $O(E/q + \sqrt{N} \cdot S)$ -subgaussian.

**Key switching** The key-switching procedure can be divided in two steps, where the first uses the secret key to generate a *public* key-switching key, and the second step consists in using the key-switching key to actually performing the key switching.

We denote by  $\tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E)$  the set of key-switching keys from a key  $z$  to a key  $s$ , having  $E$ -subgaussian noise. Given  $s$  and  $z$ , we generate  $\mathbf{K} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E)$  as  $\mathbf{K} = [\mathbf{a}_k \mid \mathbf{b}_k := \mathbf{a}_k \cdot s + \mathbf{e}_k + z \cdot \mathbf{g}] \in R_q^{d \times 2}$ , where  $\mathbf{g} = (Q_1 \cdot \hat{Q}_1, \dots, Q_d \cdot \hat{Q}_d)$  is the CRT gadget vector, that is, for  $1 \leq i \leq d$ ,  $\text{row}_i(\mathbf{K}) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot z, E)$ , where  $\Delta_i := Q_i \cdot \hat{Q}_i$ .

To key switch a ciphertext  $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(\Delta \cdot m)$ , we basically have to compute the CRT decomposition of  $a$  and multiply it by both columns of  $\mathbf{K}$ . This is shown in detail in Algorithm 3.

---

**Algorithm 3:** Key switching

---

**Input:**  $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(\Delta \cdot m, E)$  and  $\mathbf{K} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E_k)$ , both in double-CRT form.  
**Output:**  $\mathbf{c}' \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m, E')$   
**Complexity:**  $d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$   
**Noise growth:**  $E' = O(E + \sqrt{dp} \cdot D \cdot E_k)$ , where  $D = \max(D_1, \dots, D_d)$   
 $\triangleright$  Consider that  $Q = \prod_{i=1}^{\ell} q_i$  and  $D_i$  is a product of  $k$  primes, with  $\ell = k \cdot d$   
1 **for**  $1 \leq i \leq d$  **do**  
2      $\tilde{a}_i := a \bmod D_i$   
    $\triangleright$  Each base extension costs  $\ell$  NTTs and  $O(k \cdot \ell \cdot p)$  multiplications on  $\mathbb{Z}_{q_i}$   
3      $\tilde{a}_i = \text{FastBaseExtension}(\tilde{a}_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$   
    $\triangleright$  The following lines cost zero NTTs and  $O(2 \cdot d \cdot p)$  modular products  
4  $\mathbf{a} := (\tilde{a}_1, \dots, \tilde{a}_d) \in \tilde{\mathcal{R}}_Q^d$   
5  $\hat{a} = \mathbf{a} \cdot \text{col}_1(\mathbf{K})$   
6  $\hat{b} = \mathbf{a} \cdot \text{col}_2(\mathbf{K}) \triangleright \text{Noise: } O(\sqrt{dp} \cdot D \cdot E_k)\text{-subgaussian}$   
7  $\mathbf{c}' = (-\hat{a}, b - \hat{b}) \in \tilde{\mathcal{R}}_Q^2$   
8 **return**  $\mathbf{c}'$

---

## 2.9 Automorphism

Given a ciphertext  $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m)$  and an integer  $u \in \mathbb{Z}_p^*$ , we apply the Galois automorphism  $X \mapsto X^u$  to both  $a$  and  $b$ . This operation maps  $\mathbf{c}$  to another RLWE ciphertext encrypting  $m(X^u) \bmod X^p - 1$ . It also has the side effect of changing the key to  $s(X^u)$ , thus, a key switching is needed to go back to the original key  $s(X)$ . Applying the automorphism itself is done by simply rotations of coefficients, which is essentially for free, thus, the cost and the noise growth are only due to the key-switching step. We show it in detail in Algorithm 4. We are most interested in the particular case where we encrypt an integer  $v$  as  $X^v$ , since the automorphism allows us to obtain an encryption of  $u \cdot v \bmod p$  as  $X^{uv}$ .

---

**Algorithm 4:** Automorphism

---

**Input:**  $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m, E)$ ,  $u \in \mathbb{Z}_p$ , and  $\mathbf{K} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s(X^u), E_k)$ . Both  $\mathbf{c}$  and  $\mathbf{K}$  in double-CRT form.  
**Output:**  $\mathbf{c}' \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m(X^u), E')$   
**Complexity:**  $d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$   
**Noise growth:**  $E' = O(E + \sqrt{dp} \cdot D \cdot E_k)$ , where  $D = \max(D_1, \dots, D_d)$   
1 Let  $\eta$  be the mapping  $X \mapsto X^{u \bmod p}$   
2  $(\hat{a}, \hat{b}) = (\eta(a), \eta(b)) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(X^u)(\Delta \cdot m(X^u), E)$   
3  $\mathbf{c}' = \text{KeySwt}(\hat{a}, \hat{b}, \mathbf{K}) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m(X^u), E')$   
4 **return**  $\mathbf{c}'$

---

## 2.10 Ring Packing

In [MS18], Micciancio and Sorrell present a ring packing method to transform a set of  $N$  LWE samples  $(\mathbf{a}_i, b_i := \mathbf{a}_i \cdot \mathbf{s} + e_i + \Delta \mu_i) \in \mathbb{Z}_Q^{n+1}$  into a single RLWE ciphertext encrypting  $\mu = \sum_{i=0}^{N-1} \mu_i \cdot X^i$ . To do so, they define a “packing key” composed by  $n \cdot L$  RLWE ciphertexts as

$$\mathbf{K} := (\mathbf{a}, \mathbf{b} := \mathbf{a} \cdot \mathbf{z} + \mathbf{e} + \mathbf{G} \cdot \mathbf{s}) \in \hat{\mathcal{R}}_Q^{n \cdot L \times 2}$$

where  $L := \lceil \log_B(Q) \rceil$  and  $\mathbf{G} = \mathbf{I}_n \otimes (B^0, \dots, B^{\ell-1})^T \in \mathbb{Z}^{n \cdot L \times n}$  is a gadget matrix such that  $g^{-1}(\mathbf{u})\mathbf{G} = \mathbf{u}$  for any  $\mathbf{u} \in \mathbb{Z}_Q^n$ . We recall this packing procedure in Algorithm 5. Note that it requires  $O(n \cdot L)$  multiplications on  $\hat{\mathcal{R}}_Q$ .

---

**Algorithm 5:** PackLWE Algorithm from [MS18]

---

**Input:**  $[(\mathbf{a}^{(i)}, b^{(i)})]_{i < N} \in \text{LWE}_s^Q(\Delta \cdot m_i, E)$ , packing key  $\mathbf{K} := (\mathbf{a}, \mathbf{b}) \in \hat{\mathcal{R}}_Q^{n\ell \times 2}$  with  $E_R$ -subgaussian error.  
**Output:**  $(a, b) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(\Delta \cdot m)$   
**Complexity:**  $O(n \cdot N \log(Q) \log(N))$  multiplications over  $\mathbb{Z}_Q$   
**Noise growth:**  $(E, E_R) \mapsto O(\sqrt{N} \cdot E + \sqrt{n \cdot N \cdot \log(Q)} \cdot B \cdot E_R)$   
1 **for**  $0 \leq i < n$  **do**  
2      $\bar{a}_i := \sum_{j=0}^{N-1} a_i^{(j)} \cdot X^j$   
3  $\bar{\mathbf{a}} = (\bar{a}_0, \dots, \bar{a}_{n-1}) \in \hat{\mathcal{R}}_Q^n$   
4  $\bar{b} = \sum_{i=0}^{N-1} b^{(i)} \cdot X^i$   
5  $\mathbf{u} = g^{-1}(\bar{\mathbf{a}}) \in \hat{\mathcal{R}}_Q^{n \cdot L}$   
6  $(a, b) = (-\mathbf{u} \cdot \mathbf{a}, \bar{b} - \mathbf{u} \cdot \mathbf{b}) \in \hat{\mathcal{R}}_Q^2$

---

## 3 Double-CRT GSW encryption scheme

In this section, we propose an RNS-version of the GSW scheme supporting all the standard operations, like external product and homomorphic multiplication. We also present two key switching algorithms for GSW, making it possible to evaluate automorphisms on GSW ciphertexts. Moreover, we also include a new operation, which we call *shrinking*.

We present our scheme over the circulant ring  $\tilde{\mathcal{R}} := \mathbb{Z}[X]/\langle X^p - 1 \rangle$ , where  $p$  is prime, and base its security on the circulant-RLWE problem, which is as secure as the standard RLWE problem over

the prime-order cyclotomic polynomial  $\Phi_p(X) = X^{p-1} + X^{p-2} + \dots + 1$ . Moreover, since our main goal is to use the GSW scheme to run the amortized bootstrapping, we just define the encryption function to powers of  $X$  and do not present the decryption. We stress that is trivial to adapt our scheme to the usual RLWE problem using power-of-two cyclotomic rings and encrypting other types of messages.

We define the GSW ciphertexts in a more general way, by including a correction factor  $\alpha \in \mathbb{Z}$ , which is introduced by the shrinking operation and has to be removed during decryption. In more detail, the set of GSW encryptions of a message  $m$  with a scaling factor  $\alpha$  is denoted by  $\tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m)$ . Any element of this set has the form

$$\mathbf{C} = [\mathbf{a} \mid \mathbf{a} \cdot s + \mathbf{e}] + m \cdot \mathbf{G}_\alpha \in \tilde{\mathcal{R}}_Q^{2d \times 2},$$

where  $s \in \tilde{\mathcal{R}}$  is the secret key,  $\mathbf{e} \in \tilde{\mathcal{R}}^{2d}$  is the noise term, and  $\mathbf{G}_\alpha$  is the scaled gadget matrix, as described in Section 2.6. We can write  $\tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E)$  to specify that  $\mathbf{e}$  is  $E$ -subgaussian.

When we say that  $\mathbf{C}$  is in double-CRT form, we mean that each entry  $c_{i,j}$  is stored as  $\mathbf{Mat}(c_{i,j})$ , as described in Section 2.5.

- **GSW.ParamGen**( $1^\lambda$ ): Choose a prime number  $p$ , standard deviations  $\sigma_{\text{err}}, \sigma_{\text{sk}} \in \mathbb{R}$ , and an integer  $Q := \prod_{i=1}^\ell q_i$ , where  $q_1, \dots, q_\ell$  are small primes (say, with 32 bits), such that the  $(p, Q, \sigma_{\text{err}}, \sigma_{\text{sk}})$ -RLWE problem offers us  $\lambda$  bits of security. Moreover,  $p$  and  $Q$  must be coprime. Let  $d \in \mathbb{N}$  be the “number of CRT digits”. For simplicity, assume that  $d \mid \ell$  and let  $u := \ell/d \in \mathbb{Z}$ . Then, for  $1 \leq i \leq d$ , define each “CRT digit” as  $D_i := \prod_{j=(i-1) \cdot u + 1}^{i \cdot u} q_j$ , that is, a product of  $u$  consecutive primes.  
Output  $\text{params} = (p, Q, \sigma_{\text{err}}, \sigma_{\text{sk}}, d, \{q_i\}_{i=1}^\ell, \{D_i\}_{i=1}^d)$ .
- **GSW.KeyGen**( $\text{params}$ ): Sample  $s_0, \dots, s_{p-1}$  following a discrete Gaussian over  $\mathbb{Z}$  with parameter  $\sigma_{\text{sk}}$ . Let  $\bar{s} = \sum_{i=0}^{p-1} s_i \cdot X^i$  then project  $\bar{s}$  as  $s := L((1 - X)\bar{s}) \in \tilde{\mathcal{R}}$ . Output  $\text{sk} := (s, \bar{s})$ .
- **GSW.Enc**( $\mu, \text{sk}$ ): To encrypt  $\mu \in \mathbb{Z}_p$ , generate a matrix  $\mathbf{V} \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  where each row is a sample from the Circulant-LWE distribution with secret  $s$  and noise terms following a discrete Gaussian with parameter  $\sigma_{\text{err}}$ . Output  $\mathbf{V} + X^\mu \cdot \mathbf{G} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^\mu)$ .

**Lemma 1 (Security of GSW).** *If the decisional  $(p, Q, \sigma_{\text{err}}, \sigma_{\text{sk}})$ -RLWE problem is hard, then the GSW scheme over the circulant ring  $\tilde{\mathcal{R}}$  is CPA-secure for messages of the form  $X^k$ .*

*Proof.* Lemma 4 of [BDF18]. □

We now present the homomorphic operations that can be performed with GSW. Apart from the usual external product and GSW multiplication, we also define a new operation, which we can shrinking and whose main purpose is to reduce the size of the ciphertexts so that the next homomorphic operations become cheaper.

### 3.1 Shrinking gadget matrices

As explained in Section 2.6, let  $q_1, \dots, q_\ell$  be prime numbers,  $Q := \prod_{i=1}^\ell q_i$ , and  $D_1, \dots, D_d$  be the digits of the CRT basis. Define  $Q_i := Q/D_i \in \mathbb{Z}$  and  $\hat{Q}_i := (Q/D_i)^{-1} \bmod D_i$ . Also, let  $\alpha \in \mathbb{Z}_Q$



and  $\alpha_i := \alpha \bmod D_i$ . Then, the scaled gadget matrix is

$$\mathbf{G}_\alpha = \begin{bmatrix} Q_1 \cdot \hat{Q}_1 \cdot \alpha_1 & 0 \\ \vdots & 0 \\ Q_d \cdot \hat{Q}_d \cdot \alpha_d & 0 \\ 0 & Q_1 \cdot \hat{Q}_1 \cdot \alpha_1 \\ \vdots & \vdots \\ 0 & Q_d \cdot \hat{Q}_d \cdot \alpha_d \end{bmatrix} \in \mathbb{Z}^{2d \times 2}.$$

Notice that each CRT digit  $D_i$  defines two rows of  $\mathbf{G}_\alpha$ . Ideally, we would choose  $k$  digits, say,  $D_1, \dots, D_k$ , remove the two  $2k$  rows corresponding to them, and obtain a new gadget matrix with respect to the digits  $D_{k+1}, \dots, D_d$ . However, by doing so, we obtain a scaled gadget matrix  $\mathbf{G}_\beta$  with respect to a new scaling factor  $\beta$ .

For this shrinking operation, we define the projection  $\pi_k : \mathcal{R}_Q^{2d \times 2} \rightarrow \mathcal{R}_Q^{2(d-k) \times 2}$  as the function that takes a matrix  $\mathbf{C}$  and outputs  $\mathbf{C}'$  such that for  $1 \leq i \leq d-k$ ,  $\text{row}_i(\mathbf{C}') := \text{row}_{i+k}(\mathbf{C})$  and  $\text{row}_{d-k+i}(\mathbf{C}') := \text{row}_{i+d+k}(\mathbf{C})$ . Essentially, we just have to apply  $\pi_k$ , divide the result by  $D^{(k)} := D_1 \cdot \dots \cdot D_k$ , and compute the new scaling factor  $\beta$ . This procedure is shown in detail in Algorithm 6. In Lemma 2, we prove its correctness.

---

**Algorithm 6:** Shrink matrix

---

**Input:**  $\mathbf{C} \in \mathcal{R}_Q^{2d \times 2}$ , CRT digits  $D_1, \dots, D_d$ , a scaling factor  $\alpha$ , and  $k \in \mathbb{Z}$  such that  $1 \leq k < d$ .

**Output:**  $\mathbf{C}' \in \mathcal{R}_{Q'}^{2(d-k) \times 2}$  and  $\alpha' \in \mathbb{Z}$ .

- 1  $D^{(k)} := D_1 \cdot \dots \cdot D_k$
  - 2  $Q' := Q / D^{(k)}$
  - 3  $\bar{\mathbf{C}} := \pi_k(\mathbf{C})$
  - 4  $\mathbf{C}' := \bar{\mathbf{C}} / D^{(k)} \bmod Q'$
  - 5  $\alpha' := \alpha \cdot \text{CRT}_{D_{k+1}, \dots, D_d}(D^{(k)}, \dots, D^{(k)})^{-1} \bmod Q'$ .
  - 6 **return**  $\mathbf{C}', \alpha'$
- 

**Lemma 2.** Let  $Q := \prod_{i=1}^d D_i$  for coprime  $D_i$ 's and  $k$  be an integer such that  $1 \leq k < d$ . Define  $Q' := Q / (D_1 \cdot \dots \cdot D_k)$ . Then, given a scaled gadget matrix  $\mathbf{G}_\alpha$  with respect to the CRT basis  $D_1, \dots, D_d$ , Algorithm 6 outputs  $\mathbf{G}' \in \mathbb{Z}^{2(d-k) \times 2}$  and  $\alpha' \in \mathbb{Z}$  such that  $\text{CRT}^{-1}(a, b) \cdot \mathbf{G}' = \alpha' \cdot (a, b) \bmod Q'$ , for any  $(a, b) \in \mathcal{R}_{Q'}^2$ , where  $\text{CRT}^{-1}$  is the decomposition with respect to  $D_{k+1}, \dots, D_d$ .

*Proof.* Let  $Q'_i := Q' / D_i \in \mathbb{Z}$  and  $\hat{Q}'_i := (Q' / D_i)^{-1} \bmod D_i$ , for  $k+1 \leq i \leq d$ . We want to write  $\mathbf{G}'$  in terms of  $Q'_i$  and  $\hat{Q}'_i$  to show that it is indeed a scaled gadget matrix with respect to the CRT digits  $D_{k+1}, \dots, D_d$  and the modulus  $Q'$ .

First, notice that

$$\bar{\mathbf{G}} := \pi_k(\mathbf{G}) = \begin{bmatrix} Q_{k+1} \cdot \hat{Q}_{k+1} \cdot \alpha_{k+1} & 0 \\ \vdots & 0 \\ Q_d \cdot \hat{Q}_d \cdot \alpha_d & 0 \\ 0 & Q_{k+1} \cdot \hat{Q}_{k+1} \cdot \alpha_{k+1} \\ \vdots & \vdots \\ 0 & Q_d \cdot \hat{Q}_d \cdot \alpha_d \end{bmatrix} \in \mathbb{Z}^{2(d-k) \times 2}.$$

But we see that for  $k+1 \leq i \leq d$ ,

$$Q_i = Q/D_i = (Q'/D_i) \cdot D^{(k)} = Q'_i \cdot D^{(k)}$$

and

$$\hat{Q}_i = [(Q/D_i)^{-1}]_{D_i} = [(Q'/D_i)^{-1} \cdot (D^{(k)})^{-1}]_{D_i} = \hat{Q}'_i \cdot (D^{(k)})^{-1} \bmod D_i.$$

Therefore, by defining  $\alpha'_i := \alpha_i \cdot (D^{(k)})^{-1} \bmod D_i$ , we have

$$\mathbf{G}' := \frac{\pi_k(\mathbf{G})}{D^{(k)}} = \begin{bmatrix} Q'_{k+1} \cdot \hat{Q}'_{k+1} \cdot \alpha'_{k+1} & 0 \\ \vdots & 0 \\ Q'_d \cdot \hat{Q}'_d \cdot \alpha'_d & 0 \\ 0 & Q'_{k+1} \cdot \hat{Q}'_{k+1} \cdot \alpha'_{k+1} \\ \vdots & \vdots \\ 0 & Q'_d \cdot \hat{Q}'_d \cdot \alpha'_d \end{bmatrix} \in \mathbb{Z}^{2(d-k) \times 2}.$$

Now, let  $\mathbf{y} := \text{CRT}^{-1}(a, b)$ . To show that  $\mathbf{y} \cdot \mathbf{G}' = \alpha' \cdot (a, b) \bmod Q'$ , notice that the product by the first column of  $\mathbf{G}'$ , modulo  $Q'$ , gives us

$$\begin{aligned} \mathbf{y} \cdot \text{col}_1(\mathbf{G}') &= \sum_{i=k+1}^d (Q'_i \cdot \hat{Q}'_i) \cdot \alpha'_i \cdot [a]_{D_i} \\ &= \text{CRT}([\alpha' \cdot a]_{D_{k+1}}, \dots, [\alpha' \cdot a]_{D_d}) \\ &= \text{CRT}([(D^{(k)})^{-1} \cdot \alpha]_{D_{k+1}}, \dots, [(D^{(k)})^{-1} \cdot \alpha]_{D_d}) \cdot \text{CRT}([a]_{D_{k+1}}, \dots, [a]_{D_d}) \\ &= \alpha' \cdot a \end{aligned}$$

The same argument shows that  $\mathbf{y} \cdot \text{col}_2(\mathbf{G}') = \alpha' \cdot b \bmod Q'$ . □

### 3.2 Shrinking a ciphertext

Let  $\mathbf{C} = [\mathbf{a} \mid \mathbf{a} \cdot s + \mathbf{e}] + m \cdot \mathbf{G}_\alpha \in \tilde{\mathcal{R}}^{2d \times 2}$  be an encryption of  $m$  with scaling factor  $\alpha$ . We define the operation  $\text{GSW.Shrink}(\mathbf{C}, k)$  essentially by applying Algorithm 6 to  $\mathbf{C}$ , except that dividing  $\bar{\mathbf{C}} \in \tilde{\mathcal{R}}_Q^{2(d-k) \times 2}$  by  $D^{(k)}$  is done by applying the modulus switching to every row of  $\bar{\mathbf{C}}$ . This is necessary because we are assuming that  $\mathbf{C}$  is stored in double-CRT form. We show this procedure in detail in Algorithm 7 and prove its correctness in Lemma 3.

---

**Algorithm 7:** Shrink ciphertext

---

**Input:**  $\mathbf{C} \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  in double-CRT form, scaling factor  $\alpha$ , CRT digits  $D_1, \dots, D_d$ , and  $k \in \mathbb{Z}$  such that  $1 \leq k < d$ .  
**Output:**  $\mathbf{C}' \in \tilde{\mathcal{R}}_{Q'}^{2(d-k) \times 2}$  and new correction factor  $\alpha' \in \mathbb{Z}$ .  
**Complexity:**  $4 \cdot (d-k) \cdot \ell$  NTTs and  $O(k \cdot \ell^2 \cdot p)$  multiplications on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E \mapsto O(E/D^{(k)}) + \sqrt{p} \cdot S$

- 1  $D^{(k)} := D_1 \cdot \dots \cdot D_k$
- 2  $Q' := Q/D^{(k)}$
- 3  $\bar{\mathbf{C}} := \pi_k(\mathbf{C}) \in \tilde{\mathcal{R}}_{Q'}^{2(d-k) \times 2}$
- 4 **for**  $1 \leq i \leq 2 \cdot (d-k)$  **do**
- 5      $\mathbf{c}_i := \text{ModSwt}_{Q \rightarrow Q'}(\text{row}_i(\bar{\mathbf{C}}))$
- 6 Define  $\mathbf{C}'$  such that  $\text{row}_i(\mathbf{C}') = \mathbf{c}_i$ .
- 7  $\beta := \text{CRT}_{D_{k+1}, \dots, D_d}(D^{(k)}, \dots, D^{(k)})^{-1} \bmod Q'$ .
- 8  $\alpha' := \alpha \cdot \beta \bmod Q'$
- 9 **return**  $\mathbf{C}', \alpha'$

---

**Lemma 3 (Correctness and cost of ciphertext shrinking).** *Let  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E)$ ,  $k \in \mathbb{N}^*$  such that  $k < d$ , and  $\mathbf{C}', \alpha'$  be the output of Algorithm 7. Assume that  $s$  is  $S$ -subgaussian for some  $S$ . Then,  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha' \cdot m, E')$  with  $E' = O(E/D^{(k)}) + \sqrt{p} \cdot S$ .*

*Moreover, assuming that each CRT digit  $D_i$  is a product of  $\ell/d$  primes, the cost of Algorithm 7 is  $4 \cdot (d-k) \cdot \ell$  NTTs and  $O(k \cdot \ell^2 \cdot p)$  multiplications on  $\mathbb{Z}_{q_i}$ .*

*Proof.* Firstly notice that  $\bar{\mathbf{C}} = \pi_k(\mathbf{C}) = \pi_k([\mathbf{a} \mid \mathbf{a} \cdot s + \mathbf{e}]) + m \cdot \pi_k(\mathbf{G}_\alpha) = [\bar{\mathbf{a}} \mid \bar{\mathbf{a}} \cdot s + \bar{\mathbf{e}}] + m \cdot D^{(k)} \cdot \mathbf{G}_{\alpha'}$ , where  $\bar{\mathbf{a}} = \pi_k(\mathbf{a})$  and  $\bar{\mathbf{e}} = \pi_k(\mathbf{e})$ . Thus, each row of  $\bar{\mathbf{C}}$  can be seen as an RLWE sample encrypting  $\Delta \cdot \mu$ , where  $\mu \in \{m, -s\}$  and  $\Delta = D^{(k)} \cdot Q'_i \cdot \hat{Q}'_i \cdot \alpha'_i$  for some  $i$ . Thus,  $\text{ModSwt}_{Q \rightarrow Q'}(\text{row}_i(\bar{\mathbf{C}}))$  outputs an RLWE sample encrypting  $\Delta \cdot \mu / D^{(k)} = Q'_i \cdot \hat{Q}'_i \cdot \alpha'_i \cdot \mu$ . Grouping these rows to define  $\mathbf{C}'$  gives us  $\mathbf{C}' = [\mathbf{a}' \mid \mathbf{a}' \cdot s + \mathbf{e}'] + m \cdot \mathbf{G}_{\alpha'}$ .

To analyze the error growth, first notice that  $\pi_k$  does not increase the noise, thus,  $\bar{\mathbf{e}}$  is  $E$ -subgaussian. Then, the final noise  $\mathbf{e}'$  is just provenient from the modulus switching over RLWE samples with  $E$ -subgaussian noise, thus, we have  $E' = O(E/D^{(k)}) + \sqrt{p} \cdot S$ .

Over the ring  $\tilde{\mathcal{R}}_Q$ , one modulus switching to remove  $u$  primes out of  $\ell$  costs  $2\ell$  NTTs and  $O(u \cdot \ell \cdot p)$  products modulo  $q_i$ . Since we are removing  $D^{(k)}$ , which is a product of  $k\ell/d$  primes, and we execute modulus switching  $2(d-k)$  times, in total we need  $4(d-k)\ell$  NTTs and  $O(2(d-k) \cdot k \cdot \ell^2 \cdot p/d) = O(k \cdot \ell^2 \cdot p)$  multiplications on  $\mathbb{Z}_{q_i}$ .  $\square$

### 3.3 External product in RNS representation

The external product is a homomorphic multiplication between an RLWE ciphertext and a GSW ciphertext. While this is a fairly well known operation, the introduction of the RNS representation and of the scaling factor bring a few considerations to light that we must address. We present this operation in detail in Algorithm 8 and prove its correctness in Lemma 4.

**Lemma 4 (Correctness and cost of external product).** *On input  $\mathbf{c} \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m_0, E_0)$  and  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m_1, E_1)$ , Algorithm 8 outputs  $\mathbf{c}' \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m_0 \cdot m_1, E')$  where  $E' = O(\sqrt{dp} \cdot D \cdot E_1 + E_0)$  if  $m_1$  is a power of  $X$  and  $E' = O(\sqrt{dp} \cdot D \cdot E_1 + \sqrt{p} \cdot E_0 \cdot \|m_1\|)$  otherwise.*

*Moreover, it requires  $2 \cdot d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .*

---

**Algorithm 8:** RNS-friendly External product

---

**Input:**  $\mathbf{c} \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m_0, E_0)$  and  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m_1, E_1)$  both in double-CRT form.  
**Output:**  $\mathbf{c}' \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta \cdot m_0 \cdot m_1, E')$   
**Complexity:**  $2 \cdot d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E' \in O(\sqrt{dp} \cdot D \cdot E_1 + E_0)$  if  $m_1 = X^u$  for some  $u$  and  
 $E' \in O(\sqrt{dp} \cdot D \cdot E_1 + \sqrt{p} \cdot E_0 \cdot \|m_1\|)$  otherwise

- 1 Denote  $\mathbf{c} = (a, b)$ 
  - ▷ Almost for free in double-CRT form. Just group entries corresponding to prime factors of each  $D_i$  and multiply them by  $\alpha^{-1}$
- 2 **for**  $1 \leq i \leq d$  **do**
- 3     Let  $u_i := \alpha^{-1} \cdot a \bmod D_i$
- 4     Let  $u_{i+d} := \alpha^{-1} \cdot b \bmod D_i$
- 5 **for**  $1 \leq i \leq d$  **do**
- 6     Let  $v_i := \text{FastBaseExtension}(u_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$
- 7     Let  $v_{i+d} := \text{FastBaseExtension}(u_{i+d}, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$
- 8 Let  $\mathbf{v} := (v_1, \dots, v_{2d}) \in \tilde{\mathcal{R}}_Q^{2d}$  ▷ it is already in double-CRT format
- 9  $\mathbf{c}' = \mathbf{v} \cdot \mathbf{C} \in \tilde{\mathcal{R}}_Q^2$
- 10 **return**  $\mathbf{c}'$

---

*Proof.* Remember that we can write  $\mathbf{C} = [\mathbf{a} \mid \mathbf{a} \cdot s + \mathbf{e}] + m \cdot \mathbf{G}_\alpha \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  for some  $\mathbf{a} \in \tilde{\mathcal{R}}_Q^{2d}$  and  $\mathbf{e}$  is  $E$ -subgaussian. Let  $\mathbf{u} = (u_1, \dots, u_{2d})$  be the vector defined by the first loop. It is easy to see that  $\mathbf{u} = G^{-1}(\alpha^{-1} \cdot \mathbf{c})$ . Therefore, it holds that  $\mathbf{u} \cdot \mathbf{G}_\alpha = \mathbf{c} \bmod Q$ .

Applying the fast base extension to  $u_i$  outputs  $v_i = u_i + w_i$  where  $\|w_i\| \leq D_i/2$ . Thus, by defining  $D := \max(D_1, \dots, D_d)$ , we have  $\mathbf{v} = \mathbf{u} + \mathbf{w}$  where  $\|\mathbf{w}\| \leq D/2$ .

Firstly, we claim that  $\mathbf{w} \cdot \mathbf{G}_\alpha = (0, 0) \bmod Q$ . Indeed, remember that  $Q_i := Q/D_i$ ,  $\hat{Q}_i := Q_i^{-1} \bmod D_i$ ,  $\mathbf{g} = (Q_1 \cdot \hat{Q}_1 \cdot [\alpha]_{D_1}, \dots, Q_d \cdot \hat{Q}_d \cdot [\alpha]_{D_d})$  and  $\mathbf{G}_\alpha = \begin{pmatrix} (\mathbf{g} \mathbf{0})^T & (\mathbf{0} \mathbf{g})^T \end{pmatrix} \in \mathbb{Z}^{2d \times 2}$ , thus, by writing  $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2)$  we can see that  $(\mathbf{w}_1, \mathbf{w}_2) \cdot \mathbf{G}_\alpha = (\mathbf{w}_1 \cdot \mathbf{g}, \mathbf{w}_2 \cdot \mathbf{g})$ . But for both  $i = 1, 2$ , we have

$$\mathbf{w}_i \cdot \mathbf{g} = \sum_{j=1}^d (w'_{i,j} \cdot D_j) \cdot (Q_j \cdot \hat{Q}_j \cdot [\alpha]_{D_i}) = \sum_{j=1}^d w'_{i,j} \cdot Q \cdot \hat{Q}_j \cdot [\alpha]_{D_i} = 0 \bmod Q.$$

Therefore, modulo  $Q$ , it holds that

$$\begin{aligned} \mathbf{c}' &:= \mathbf{v} \cdot \mathbf{C} \\ &= \mathbf{v} \cdot ([\mathbf{a}, \mathbf{a} \cdot s + \mathbf{e}] + \mathbf{G}_\alpha \cdot m_1) \\ &= [\mathbf{v} \cdot \mathbf{a}, \mathbf{v} \cdot \mathbf{a} \cdot s + \mathbf{v} \cdot \mathbf{e}] + \mathbf{v} \cdot \mathbf{G}_\alpha \cdot m_1 \\ &= [\mathbf{v} \cdot \mathbf{a}, \mathbf{v} \cdot \mathbf{a} \cdot s + \mathbf{v} \cdot \mathbf{e}] + \mathbf{u} \cdot \mathbf{G}_\alpha \cdot m_1 \\ &= [\mathbf{v} \cdot \mathbf{a}, \mathbf{v} \cdot \mathbf{a} \cdot s + \mathbf{v} \cdot \mathbf{e}] + [a, a \cdot s + e + \Delta \cdot m_0] \cdot m_1 \\ &= [\underbrace{\mathbf{v} \cdot \mathbf{a} + a \cdot m_1}_{a'}, \underbrace{a' \cdot s + \mathbf{v} \cdot \mathbf{e} + e \cdot m_1 + \Delta \cdot m_0 \cdot m_1}_{e'}]. \end{aligned}$$

Hence,  $\mathbf{c}$  is indeed an RLWE encryption of  $m_0 \cdot m_1$ .

Moreover, since  $\|\mathbf{v}\| \leq D$ , we have that  $\mathbf{v} \cdot \mathbf{e}$  is  $(\sqrt{dp} \cdot D \cdot E_1)$ -subgaussian. If  $m_1 = X^z$  for some  $z$ , the product  $e \cdot m_1 \bmod X^p - 1$  just rotates the coefficients of  $e$ , but do not change the distribution, thus,  $e'$  is  $(\sqrt{dp} \cdot D \cdot E_1 + E_0)$ -subgaussian. In general,  $e \cdot m_1$  is  $(\sqrt{p} \cdot E_0 \cdot \|m_1\|)$ -subgaussian, and  $e'$  is  $(\sqrt{dp} \cdot D \cdot E_1 + \sqrt{p} \cdot E_0 \cdot \|m_1\|)$ -subgaussian.

It remains to analyze the cost of the algorithm. The first loop costs only  $2\ell$  modular multiplications. Since each CRT digit  $D_i$  has  $\ell/d$  prime factors, each base extension costs  $\ell$  NTTs and  $O((\ell/d) \cdot (\ell - \ell/d) \cdot p)$  modular multiplications. Thus, the second loop costs  $2 \cdot d \cdot \ell$  NTTs and  $O(\ell^2 \cdot (1 - 1/d) \cdot p) = O(\ell^2 \cdot p)$  modular multiplications. Since the output of **FastBaseExtension** is already in double-CRT format, the product  $\mathbf{v} \cdot \mathbf{C}$  does not require any NTT and is performed via pointwise multiplication, thus, it costs  $4d\ell p$  modular multiplications. Therefore, the total cost is  $2 \cdot d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  $\square$

### 3.4 Homomorphic multiplication

This operation<sup>3</sup> takes GSW encryptions of two messages and output a GSW encryption of their product. This is done by performing one external product for each row of one of the ciphertexts, therefore, the cost is exactly  $2 \cdot d$  times the cost of one external product. Since each row is multiplied independently, the noise growth is the same as in the external product. We show it in detail in Algorithm 9. The only caveat is the scaling factor of the output. Both input ciphertexts have scaling factors, say  $\alpha_0$  and  $\alpha_1$ , so one could expect that the output would be scaled by  $\alpha_0 \cdot \alpha_1$ . However, since external products output RLWE encryptions that do not depend on the scaling factor of the GSW ciphertext, the output of the GSW multiplication is scaled only by, say,  $\alpha_0$ .

---

#### Algorithm 9: RNS-friendly GSW homomorphic multiplication

---

**Input:**  $\mathbf{C}_0 \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha_0 \cdot m_0, E_0)$  and  $\mathbf{C}_1 \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha_1 \cdot m_1, E_1)$  both in double-CRT form.  
**Output:**  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d((\alpha_0) \cdot m_0 \cdot m_1, E)$   
**Complexity:**  $4 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E \in O(\sqrt{dp} \cdot D \cdot E_1 + E_0)$  if  $m_1 = X^u$  for some  $u$  and  $E' \in O(\sqrt{dp} \cdot D \cdot E_1 + \sqrt{p} \cdot E_0 \cdot \|m_1\|)$  otherwise  
 $\triangleright$  Consider  $\Delta_i := Q_i \cdot \hat{Q}_i \cdot \alpha_0 \bmod D_i$   
1 **for**  $1 \leq i \leq d$  **do**  
2     Let  $\mathbf{c}_i := \text{row}_i(\mathbf{C}_0) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot m_0, E_0)$   
3      $\mathbf{c}'_i = \mathbf{c}_i \boxtimes \mathbf{C}_1 \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot m_0 \cdot m_1, E)$   
4 **for**  $d+1 \leq i \leq 2 \cdot d$  **do**  
5     Let  $\mathbf{c}_i := \text{row}_i(\mathbf{C}_0) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot m_0 \cdot s, E_0)$   
6      $\mathbf{c}'_i = \mathbf{c}_i \boxtimes \mathbf{C}_1 \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot m_0 \cdot m_1 \cdot s, E)$   
7 Let  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  such that  $\text{row}_i(\mathbf{C}') = \mathbf{c}'_i$   
8 **return**  $\mathbf{C}'$

---

### 3.5 Key switching for GSW

In this section we present two methods to switch the key of GSW ciphertexts. They represent a memory-noise tradeoff, as the first method requires less key material, but introduces more noise than the second one. These key-switching algorithms allow us to use Galois automorphisms on GSW ciphertexts and give us a simple and efficient algorithm to perform the pointwise multiplication needed in the beginning of the bootstrapping.

<sup>3</sup> In [CGGI20], a homomorphic multiplication between two GSW ciphertexts is called *internal product*.

**Encrypting non-powers of  $X$  under the CLWE problem:** If one instantiates our double-CRT GSW scheme over the usual power-of-two cyclotomic ring, i.e., modulo  $X^N + 1$ , then there is no security issue and one can freely choose the keys that will be switched. However, because we are using the circulant ring, defined modulo  $X^p - 1$ , we have to be more careful.

In [BDF18], it was proved that the GSW scheme over circulant rings is secure if one just encrypts powers of  $X$ . But to key switch from a secret key  $s \in \tilde{\mathcal{R}}$  to another secret key  $z \in \tilde{\mathcal{R}}$ , we have to encrypt  $z$ , which is not a power of  $X$ . Thus, we extend the results of [BDF18] to show that it is also safe to encrypt  $z$ .

The main idea is the following: using a circular ring introduces a security issue because an attacker can interpret an element  $a \in \tilde{\mathcal{R}}$  as a polynomial  $a' \in \mathbb{Z}[X]$ . In this case, it holds that  $a' = a + u \cdot (X^p - 1)$  for some  $u \in \mathbb{Z}[X]$ . Then, evaluating  $a'$  at one yields  $a'(1) = a(1) \in \mathbb{Z}$ . Thus, taking a ciphertext  $(a, b := a \cdot s + e)$  defined modulo  $X^p - 1$  and evaluating it at one would produce the pair  $(a(1), b(1) = a(1) \cdot s(1) + e(1)) \in \mathbb{Z}^2$ , which could leak information about  $s$ .

Bonnoron, Ducas, and Fillinger [BDF18] prove that the Circulant LWE (CLWE) is as hard as the RLWE by applying to the RLWE samples a function that fixes the values of the polynomials when they are evaluated at one, such that  $(a(1), b(1))$  is always equal to zero, thus, independent of the secret key. Then, adding messages of the form  $m(X) = X^k$  to the CLWE samples produces ciphertexts of the form  $(a, b + \Delta \cdot m) \in \tilde{\mathcal{R}}^2$  that also have a fixed value when evaluated at one, namely,  $(0, \Delta)$ . Therefore, they leak no information about the message. The crucial property here is that  $m(1) = 1$  for any  $k$ . But we notice that one can use a more general property, namely, if there is a constant  $c$  such that for each message  $m$ , it holds that  $m(1) = c$ , then evaluating the ciphertexts at one produces  $(0, \Delta \cdot c)$ , which is still independent of the messages. Thus, we can generalize the results of [BDF18] to argue that the GSW over circulant rings is CPA-secure for any messages  $m$ , as long as  $m(1)$  is already known.

In the case of the key-switching keys, we are only encrypting the secret polynomials of the CLWE problem, but they always result in zero when they are evaluated at one, because they are always of the form  $s = L((1 - X) \cdot s')$ , as explained in Section 2.2. Therefore, it is secure to encrypt them under the CLWE problem. We prove this result formally in Appendix A.

**GSW key switching via two-layer reconstruction** Given  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_z^d(\alpha \cdot m)$ , remember that for  $1 \leq i \leq d$ ,  $\text{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(\Delta_i \cdot m)$ , where  $\Delta_i := Q_i \cdot \hat{Q}_i \cdot \alpha_i$ . Also, for  $d + 1 \leq i \leq 2 \cdot d$ ,  $\text{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(-\Delta_i \cdot m \cdot z)$ . Thus, to switch  $\mathbf{C}$  to a key  $s$ , we just need to use the first  $d$  rows. Namely, we use the RLWE key switching to obtain RLWE encryptions of  $\Delta_i \cdot m$  under  $s$ , then we multiply these ciphertexts by  $-s$  to construct the last  $d$  rows. This procedure is shown in detail in Algorithm 10.

From the costs of the external product and of the key switching, algorithms 8 and 3, we see that we need  $3 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ . From the noise growth presented in these algorithms, we see that after key switching, the noise is  $\hat{E}$ -subgaussian, where  $\hat{E} \in O(E + \sqrt{dp} \cdot D \cdot E_k)$  and  $D = \max(D_1, \dots, D_d)$ . Thus, after the external products, we have an  $E'$ -subgaussian, where  $E' \in O(\sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \hat{E} \cdot \|s\|) = O(\sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E + p \cdot \sqrt{d} \cdot D \cdot E_k \cdot \|s\|)$

**Noise-reduced GSW key switching via parallel reconstruction** When we key switching the first  $d$  rows from  $z$  to  $s$ , we generate RLWE samples with larger noise. Then, when we use these samples to reconstruct the other  $d$  rows, we accumulate more noise over them, generating thus samples with even larger noise, proportional to  $p$ .



---

**Algorithm 10:** GSW key switching

---

**Input:**  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_z^d(\alpha \cdot m, E)$ ,  $\mathbf{K} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E_k)$ ,  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot (-s), E_s)$ , all in double-CRT form.  
**Output:**  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E')$ ,  
**Complexity:**  $3 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E' \in O(\sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E + p \cdot \sqrt{d} \cdot D \cdot E_k \cdot \|s\|)$   
1 **for**  $1 \leq i \leq d$  **do**  
2      $\mathbf{c}_i = \text{KeySwt}(\text{row}_i(\mathbf{C}), \mathbf{K}) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot m)$   
3      $\mathbf{c}_{d+i} = \mathbf{c}_i \boxplus \mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot m \cdot s)$   
4 Let  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  such that  $\text{row}_i(\mathbf{C}') = \mathbf{c}_i$   
5 **return**  $\mathbf{C}'$

---

The key switching that we present in this section avoids that “double accumulation” by producing independent RLWE samples that can be subtracted to reconstruct the remaining rows. Because subtraction increases the noise linearly, we accumulate less noise in the last  $d$  rows of the GSW ciphertext. At the end, we save a factor  $\sqrt{p}$  in the final noise. For this, we need an extra key encrypting the  $s \cdot z$  and we replace the GSW encryption of  $s$  by a key-switching key from  $s$  to  $s$  itself. We present this procedure in detail in Algorithm 11

---

**Algorithm 11:** NoiseReducedGSWKeySwt

---

**Input:**  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_z^d(\alpha \cdot m, E)$ ,  $\mathbf{K}_z \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E_z)$ ,  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s, E_s)$ , and  $\mathbf{K}_{sz} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s \cdot z, E_{sz})$ , all in double-CRT form.  
**Output:**  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E')$ ,  
**Complexity:**  $3 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E' \in O(\sqrt{dp} \cdot D \cdot E_{sz} + \sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E)$   
▷ Construct the first  $d$  rows of the GSW ciphertext  
1 **for**  $1 \leq i \leq d$  **do**  
2      $\mathbf{c}_i = \text{KeySwt}(\text{row}_i(\mathbf{C}), \mathbf{K}_z) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot m)$   
▷ Now construct the last  $d$  rows  
3 **for**  $1 \leq i \leq d$  **do**  
4     Let  $(a, b) = \text{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q \text{LWE}_z(\Delta_i \cdot m)$   
5     **for**  $1 \leq i \leq d$  **do**  
6         ▷ Each base extension costs  $\ell$  NTTs and  $O(p \cdot \ell^2 / d)$  multiplications on  $\mathbb{Z}_{q_i}$   
7          $h_i = \text{FastBaseExtension}(a \bmod D_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$   
8          $y_i = \text{FastBaseExtension}(b \bmod D_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$   
9      $\mathbf{h} := (h_1, \dots, h_d)$   
10     $\mathbf{y} := (y_1, \dots, y_d)$   
11     $a' := \mathbf{h} \cdot \text{col}_1(\mathbf{K}_{sz}) - \mathbf{y} \cdot \text{col}_1(\mathbf{K}_s)$   
12     $b' := \mathbf{h} \cdot \text{col}_2(\mathbf{K}_{sz}) - \mathbf{y} \cdot \text{col}_2(\mathbf{K}_s)$   
13     $\mathbf{c}_{d+i} := (a', b') \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot m \cdot s)$   
14 Let  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$  such that  $\text{row}_i(\mathbf{C}') = \mathbf{c}_i$   
15 **return**  $\mathbf{C}'$

---

**Lemma 5 (Correctness and cost of GSW key switching with parallel reconstruction).**  
On input  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_z^d(\alpha \cdot m, E)$ ,  $\mathbf{K}_z \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(z, E_k)$ ,  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s, E_z)$ , and  $\mathbf{K}_{sz} \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s \cdot z, E_{sz})$ .

$z, E_s)$ , Algorithm 8 outputs  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E')$  where

$$E' = O\left(\sqrt{p} \cdot (\sqrt{d} \cdot D \cdot E_{sz} + \sqrt{d} \cdot D \cdot E_s + \|s\| \cdot E)\right).$$

Moreover, it requires  $3 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .

*Proof.* The correctness of the first  $d$  rows of the output  $\mathbf{C}'$  follows from the correctness of the RLWE key switching. For the last  $d$  rows, consider the following.

Write  $\mathbf{K}_s := [\mathbf{a}_s \mid \mathbf{b}_s := \mathbf{a}_s \cdot s + \mathbf{e}_s + s \cdot \mathbf{g}] \in \tilde{\mathcal{R}}_Q^{d \times 2}$  and  $\mathbf{K} := [\mathbf{a}_k \mid \mathbf{b}_k := \mathbf{a}_k \cdot s + \mathbf{e}_k + s \cdot z \cdot \mathbf{g}] \in \tilde{\mathcal{R}}_Q^{d \times 2}$ , where  $\mathbf{g} \in \mathbb{Z}^d$  is the CRT gadget vector. Also, write  $(a, b) = \text{row}_i(\mathbf{C})$  with  $b = a \cdot z + e + \Delta_i \cdot m$ .

Then  $\mathbf{y} \cdot \text{col}_2(\mathbf{K}_s) = \mathbf{y} \cdot \mathbf{a}_s \cdot s + \mathbf{y} \cdot \mathbf{e}_s + s(a z + e + \Delta_i \cdot m) = \mathbf{y} \cdot \mathbf{a}_s \cdot s + \mathbf{y} \cdot \mathbf{e}_s + a \cdot s \cdot z + e \cdot s + \Delta_i \cdot m \cdot s$  and  $\mathbf{h} \cdot \text{col}_2(\mathbf{K}_{sz}) = \mathbf{h} \cdot \mathbf{a}_k \cdot s + \mathbf{h} \cdot \mathbf{e}_k + a \cdot s \cdot z$ . Hence,

$$b' := \mathbf{h} \cdot \text{col}_2(\mathbf{K}_{sz}) - \mathbf{y} \cdot \text{col}_2(\mathbf{K}_s) = \underbrace{(\mathbf{h} \cdot \mathbf{a}_k - \mathbf{y} \cdot \mathbf{a}_s)}_{a'} \cdot s + \underbrace{\mathbf{h} \cdot \mathbf{e}_k - \mathbf{y} \cdot \mathbf{e}_s - e \cdot s}_{e'} - \Delta_i \cdot m \cdot s.$$

By defining  $D = \max(D_1, \dots, D_d)$ , we have that  $e'$  is  $E'$ -subgaussian, where

$$E' \in O(\sqrt{dp} \cdot D \cdot E_{sz} + \sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E).$$

Thus,  $(a', b') \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot m \cdot s, E')$ .

Now, it remains to analyze the cost. The  $d$  RLWE key switchings executed in line 2 cost, in total,  $d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ . All the fast base extensions cost in total  $2 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  modular products. The remaining operations just cost  $4 \cdot d \cdot p$  modular products. Therefore, the algorithm requires  $3 \cdot d^2 \cdot \ell$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  $\square$

### 3.6 GSW automorphism

Given  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot m, E)$  and  $\eta : X \mapsto X^u$ , for some  $u \in \mathbb{Z}$ , we just have to apply  $\eta$  to each row of  $\mathbf{C}$ , then apply one of the GSW key switching algorithms described in Section 3.5.

Notice that the first  $d$  rows of  $\mathbf{C}$  are regular RLWE encryptions, thus, the automorphism works as usual producing new RLWE encryptions under the key  $\eta(s)$ . The other  $d$  rows can be seen as RLWE encryptions of the  $-\Delta_i \cdot m \cdot s$ , therefore, the automorphism generates RLWE encryptions of  $-\Delta_i \cdot \eta(m) \cdot \eta(s)$ , which correspond to the last  $d$  rows of a GSW encryption of  $\Delta_i \cdot \eta(m)$  under the key  $\eta(s)$ . Moreover,  $\eta$  does not change the distribution of the noise. In summary,  $\eta(\mathbf{C}) \in \tilde{\mathcal{R}}_Q \text{GSW}_{\eta(s)}^d(\alpha \cdot \eta(m), E)$ .

Then, applying GSW key switching on  $\eta(\mathbf{C})$  gives us  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot \eta(m), E')$ . Hence, the noise growth and the cost are same as the ones of the chosen key switching.

### 3.7 GSW evaluation of scalar products on the exponent

In this section we consider the problem of evaluating a scalar product between two vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_p^k$ , when  $\mathbf{u}$  is known in clear and each entry of  $\mathbf{v}$  is encrypted as  $X^{v_i}$  into a GSW sample. The output is a GSW encryption of  $X^{\mathbf{u} \cdot \mathbf{v} \bmod p}$ . This is a general operation and can possibly be used in several scenarios, but we are particularly interested in using it during the homomorphic evaluation of the inverse NTT in our bootstrapping algorithm.

The straightforward way of implementing this scalar product is the following: apply the automorphism  $X \mapsto X^{u_i}$  to obtain GSW encryptions of  $X^{u_i \cdot v_i \bmod p}$ , then use the GSW multiplication  $k$  times to obtain a GSW encryption of  $\prod_{i=1}^k X^{u_i \cdot v_i} = X^{\mathbf{u} \cdot \mathbf{v} \bmod p}$ . However, each GSW multiplication costs  $2d$  external products, so this naïve implementation needs  $2kd$  external products. We want to reduce that to around  $k \cdot d$ , thus, halving the cost.

Moreover, instead of using  $k$  automorphisms on GSW ciphertexts, which cost essentially  $2 \cdot d \cdot k$  RLWE key switchings, we want to use automorphisms on the RLWE samples so that the cost of  $k$  automorphisms also drops to  $k \cdot d$ , that is, halving the cost compared to GSW automorphisms.

Hence, given  $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{v_i})$ , we define a trivial and noiseless ciphertext  $\mathbf{C}'_0 \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{v_i})$ , and we want to compute for  $1 \leq i \leq k$

$$\mathbf{C}'_i = \text{Auth}(\mathbf{C}_i, u_i) \cdot \mathbf{C}'_{i-1}$$

The main idea is to extract the  $d$  rows of  $\mathbf{C}'_i$  that correspond to the RLWE samples encrypting  $X^m$  for some message  $m$  and ignore the other  $d$  rows. So, let  $\mathbf{c}_j \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_j \cdot X^m)$  be the  $j$ -th row of  $\mathbf{C}'_i$ . Instead of applying the automorphism  $u_i$  to  $\mathbf{C}'_i$ , we can use a technique from [BDF18] and apply  $u_i^{-1} \bmod p$  to  $\mathbf{c}_j$ , then multiply it with  $\mathbf{C}'_{i-1}$  via external product, and apply the automorphism  $u_i$  in the end. This gives us

1.  $\mathbf{c}'_j = \text{Auth}(\mathbf{c}_i, u_i^{-1})$  (encrypts  $X^{u_i^{-1} \cdot m}$ )
2.  $\mathbf{c}''_j = \mathbf{c}'_j \cdot \mathbf{C}_i$  (encrypts  $X^{u_i^{-1} \cdot m + v_i}$ )
3.  $\mathbf{c}'''_j = \text{Auth}(\mathbf{c}''_i, u_i)$  (encrypts  $X^{m + u_i \cdot v_i}$ )

Repeating this  $k$  times, at the end, we have  $\mathbf{c}'''_j \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_j \cdot X^{\sum u_i v_i})$ , as desired. Notice that for each  $i$ , the first and the third step are automorphisms, so can compose them and run a single key switching instead of two. Finally, repeating this  $d$  times, we obtain all the  $d$  first rows of the GSW ciphertext encrypting the  $X^{\mathbf{u} \cdot \mathbf{v}}$ , and it remains to construct the other  $d$  rows essentially by multiplying by  $-s$ , as it was done in the GSW key switching in Algorithm 10.

Since the whole algorithm uses  $k \cdot d$  external products and  $d \cdot k$  RLWE automorphisms, the total cost is  $3 \cdot k \cdot d^2 \cdot \ell$  NTTs. Notice that  $k$  GSW multiplications plus  $k$  GSW automorphisms would cost  $7 \cdot k \cdot d^2 \cdot \ell$  NTTs, so we are gaining a factor of around 2.33.

## 4 Bootstrapping

In this section, we show how we can use our circulant GSW scheme to evaluate a bootstrapping algorithm with polynomial noise overhead and sublinear number of homomorphic operations per refreshed message.

### 4.1 Homomorphic Number Theoretic Transform

With a more expressive accumulator in hand, namely, our circulant GSW scheme equipped of Galois automorphisms and being able to work over  $\mathbb{Z}_p$ , we can finally replace the homomorphic Nussbaumer transform and the SlowMult algorithm used in [MS18] by the Number Theoretic Transform (NTT) and a simpler point-wise multiplication.

It is well known that over “cyclic polynomial rings” of the form  $R_q := \mathbb{Z}_q[X]/\langle X^N - 1 \rangle$ , where  $N$  is a power of two, we can multiply two elements  $a, z \in R_p$  in time  $O(N \log N)$  by using the

---

**Algorithm 12: EvalScalarProd: Evaluate scalar product in the exponent of  $X$** 


---

**Input:**  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot (-s), E_s)$ ,  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s(X^v), E_k)$  for all  $v \in \mathbb{Z}_p$ , and for  $1 \leq i \leq k$ ,  
 $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{m_i}, E_i)$  and  $u_i \in \mathbb{Z}_p$ . All  $\mathbf{K}_v$  and  $\mathbf{C}_i$  in double-CRT form.

**Output:**  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^y, E')$  where  $y = \sum_{i=1}^k u_i \cdot m_i \bmod p$ .

**Complexity:**  $3 \cdot k \cdot d^2 \cdot \ell$  NTTs and  $O(k \cdot d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .

**Noise growth:**  $E \mapsto O\left(\sum_{i=1}^k \left(\sqrt{d} D p \cdot \|s\| \cdot E_i\right) + \sqrt{d p} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E_1 + \sqrt{p} \cdot \|s\| \cdot \sqrt{k+1} \cdot E_{KS}\right)$ ,  
where  $E_{KS} \in O(\sqrt{d p} \cdot D \cdot E_k)$

▷ Consider that  $\Delta_i := Q_i \cdot \hat{Q}_i \cdot \alpha_i$ , where  $\alpha_i = \alpha \bmod D_i$

- 1 Define  $u_{k+1} = 1$
- 2 **for**  $1 \leq i \leq d$  **do**
- 3     Let  $\mathbf{c}_i := \text{row}_i(\mathbf{C}_1) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot X^{m_1}, E_1)$
- 4      $\mathbf{c}_i = \text{Auth}(\mathbf{c}_i, u_1 \cdot u_2^{-1} \bmod p) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot X^{m_1 \cdot u_1 \cdot u_2^{-1}}, E_1 + E_{KS})$   
       ▷ Let  $S^{(j)} := u_{j+1}^{-1} \cdot \sum_{i=1}^j m_i \cdot u_i \bmod p$
- 5     **for**  $2 \leq j \leq k$  **do**
- 6          $\mathbf{c}_i = \mathbf{c}_i \boxplus \mathbf{C}_j \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot X^{S^{(j-1)} + m_j}, \sum_{i=2}^j (\sqrt{d p} \cdot D \cdot E_i) + (E_1 + E_{KS}))$
- 7          $\mathbf{c}_i = \text{Auth}(\mathbf{c}_i, u_j \cdot u_{j+1}^{-1} \bmod p) \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\Delta_i \cdot X^{S^{(j)}}, \sum_{i=1}^j (\sqrt{d p} \cdot D \cdot E_i) + E_1 + \sqrt{j+1} \cdot E_{KS})$
- ▷ Now, construct the other  $d$  rows of the GSW sample
- 8 **for**  $1 \leq i \leq d$  **do**
- 9     Let  $\mathbf{c}_{d+i} := \mathbf{c}_i \boxplus \mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{LWE}_s(-\Delta_i \cdot X^{S^{(k)}} \cdot s, E)$
- 10 Define  $\mathbf{C}' \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{S^{(k)}}, E)$  such that  $\text{row}_i(\mathbf{C}') = \mathbf{c}_i$
- 11 **return**  $\mathbf{C}'$

---

NTT. For this, assume that  $p \equiv 1 \pmod{N}$ , then there exists a primitive  $N$ -root of unity  $\omega \in \mathbb{Z}_p$ . The NTT is an algorithm that takes  $a \in R_p$ , interprets it as a polynomial in  $\mathbb{Z}_p[X]$ , and, in time  $O(N \log N)$ , outputs the vector  $(a(\omega^0), a(\omega^1), \dots, a(\omega^{N-1})) \in \mathbb{Z}_p^N$ . Now let  $\odot : \mathbb{Z}_p^N \rightarrow \mathbb{Z}_p^N$  be the entrywise multiplication. Then it holds that

$$\text{NTT}^{-1}(\text{NTT}(a) \odot \text{NTT}(z)) \equiv N \cdot a \cdot z \pmod{\langle X^N - 1, q \rangle}.$$

However, over “negacyclic polynomial rings” of the form  $\hat{\mathcal{R}}_p := \mathbb{Z}_p[X]/\langle X^N + 1 \rangle$ , to perform this multiplication, we first have to multiply the coefficients of  $a$  and  $z$  by powers of some primitive  $2N$ -root of unity  $\psi \in \mathbb{Z}_p$ , then apply the NTT and inverse NTT as usual, and finally multiply by powers of  $\psi^{-1}$ . In more detail, let  $\boldsymbol{\psi} := (\psi^0, \psi, \dots, \psi^{N-1})$  and  $\boldsymbol{\psi}^{-1} := (\psi^0, \psi^{-1}, \dots, \psi^{-(N-1)})$ , where  $\psi$  is a  $2N$ 'th root of unity as defined above, then

$$\boldsymbol{\psi}^{-1} \odot \text{NTT}^{-1}(\text{NTT}(\boldsymbol{\psi} \odot \mathbf{a}) \odot \text{NTT}(\boldsymbol{\psi} \odot \mathbf{z})) \equiv N \cdot a \cdot z \pmod{\langle X^N + 1, q \rangle}$$

Because we now need a  $2N$ -root of unity modulo  $p$ , we need  $p \equiv 1 \pmod{2N}$ . Notice that the NTT is of dimension  $N$ , not  $2N$ . In particular, given  $\psi$ , the  $N$ -th root of unity used by the NTT can be defined as  $\omega = \psi^2 \bmod p$ .

This radix- $m$  version of the NTT algorithm recursively splits the  $N$ -dimensional input into  $m$  vectors of dimension  $\frac{N}{m}$ . Then, after  $\rho$  recursive levels, we reach the base case of the recursion and we apply a quadratic algorithm to compute the NTT of inputs of size  $\frac{N}{m^\rho}$ . Typically, one sets  $\rho = \log_m(N)$ , such that the quadratic step is executed over inputs of size one and are actually void, obtaining then complexity  $O(N \cdot \log N)$ . However, because the noise overhead of the homomorphic NTT is proportional to  $N^\rho$ , we restrict ourselves to instantiating the algorithm with small values of  $\rho$  only.

---

**Algorithm 13:**  $\text{NTT}_m^{-1}$  - Inverse NTT in time  $O\left(\rho \cdot N^{1+\frac{1}{\rho}}\right)$ 


---

**Input:**  $(f_0, \dots, f_{N-1}) \in \mathbb{Z}_p^N, \rho \in \mathbb{Z}^+, \tilde{\rho} \in \mathbb{Z}^+$ , where  $\tilde{\rho}$  starts at 1,  $m \in \mathbb{Z}^+$  s.t.  $m \mid N$

**Output:**  $\text{NTT}_m^{-1}(\mathbf{f})$

```

1 if  $\rho = 1$  then
2    $\mathbf{u} = \psi^{-1} N^{-1} \bmod p$ 
3 else
4    $\mathbf{u} = (1, 1, \dots, 1) \in \mathbb{Z}^N$ 
5 if  $\tilde{\rho} = \rho$  then
6    $\triangleright$  Trivial quadratic algorithm, time  $O(N^2)$ 
7   for  $0 \leq j < N$  do
8      $a_j = \sum_{i=0}^{N-1} f_i \cdot u_j \cdot w_N^{-i \cdot j} \bmod p$ 
9 else
10   $\triangleright$  General case with recursive calls
11  for  $0 \leq i < m$  do
12     $\mathbf{g} = (f_i, f_{m+i}, \dots, f_{m(N/m-1)+i})$ 
13     $\mathbf{h}^{(i)} = \text{NTT}_m^{-1}(\mathbf{g}, \rho + 1)$ 
14    for  $0 \leq k_1 < \frac{N}{m}$  do
15      for  $0 \leq k_2 < m$  do
16         $j = k_1 + \frac{N}{m} \cdot k_2$ 
17         $a_j = \sum_{i=0}^{m-1} h_{k_1}^{(i)} \cdot u_j \cdot w_N^{-i \cdot k_1} \cdot w_m^{-i \cdot k_2} \bmod p$ 
18 return  $(a_0, \dots, a_{N-1})$ 

```

---

We show radix- $m$  inverse NTT in detail in Algorithm 13, where the multiplication by  $\psi^{-1}$  and also by the inverse of  $N$  modulo  $p$  is already included in the last step, so that the output already corresponds to the product of the two polynomials. We denote an  $k$ -th root of unity in  $\mathbb{Z}_p$  by  $w_k$ . In the beginning of the algorithm, we start with  $w_N$ , then all the others roots of unity that appears in all the recursive calls are just powers of  $w_N$ .

**Number of operations of homomorphic inverse NTT:** The time complexity of a radix- $m$  NTT is standard: the number of operations of lines 7 and 15 can be represented by  $T(N) = m \cdot T(N/m) + m \cdot N$ . Iterating it  $\rho$  times gives us  $T(N) = m^\rho \cdot T(N/m^\rho) + \rho \cdot m \cdot N$ . Finally, we reach the end of the recursion and the quadratic algorithm is used, thus, replacing  $T(N/m^\rho)$  by  $(N/m^\rho)^2$ , we have  $T(N) = N^2/m^\rho + \rho \cdot m \cdot N$ . By choosing  $m = N^{1/\rho}$ , we obtain the optimal complexity:

$$O\left(\rho \cdot N \cdot m + \frac{N^2}{m^\rho}\right) = O\left(\rho \cdot N^{1+\frac{1}{\rho}}\right) \quad (1)$$

With this, it is now easy to prove the complexity and noise overhead of the homomorphic evaluations of this algorithm.

**Lemma 6 (Time complexity in terms of homomorphic operations).** *The homomorphic evaluation of the inverse NTT, Algorithm 13, of dimension  $N$ , can be executed with  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho\right)$  homomorphic operations (GSW multiplications and automorphisms).*

*Proof.* The input of the algorithm is a vector of (circulant) GSW ciphertexts encrypting  $X^{f_i}$ . To add each term of the sums shown in lines 7, we just have to apply an automorphism, obtaining

an encryption of  $X^{f_i \cdot u_j \cdot w_n^{-i \cdot j} \bmod p}$ , and one homomorphic multiplication, to accumulate the term in the partial sum.

For the sum of line 15, we proceed in the same way, by applying the automorphism  $X \mapsto X^w$ , where  $w = u_j \cdot w_n^{-i \cdot k_1} \cdot w_m^{-i \cdot k_2}$ , then one multiplication.

Thus, in total, we have  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho\right)$  homomorphic operations.  $\square$

Since each GSW multiplication and automorphisms can be implemented with  $O(d^2 \cdot \ell)$  NTTs and  $O(d \cdot \ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$  using our double-CRT instantiation of GSW, we have the following result.

**Corollary 1 (Time complexity in terms of NTTs and modular multiplications).** *Let  $Q = \prod_{i=1}^{\ell} q_i$  be the ciphertext modulus. Let  $d$  be the number of CRT digits used in the GSW ciphertexts. Then the homomorphic evaluation of the inverse NTT, Algorithm 13, of dimension  $N$ , can be executed with  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d^2 \cdot \ell\right)$  NTTs and  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d \cdot \ell^2 \cdot p\right)$  multiplications modulo  $q_i$ .*

Finally, by assuming that each sum in lines 7 and 15 is implemented with Algorithm 12, we can have a concrete instead of asymptotic estimation of the number of NTTs.

**Lemma 7 (Number of NTTs used the homomorphic inverse NTT).** *Let  $Q = \prod_{i=1}^{\ell} q_i$  be the ciphertext modulus. Let  $d$  be the number of CRT digits used in the GSW ciphertexts. Consider the Algorithm 13 with recursive level  $\rho$ , dimension  $N$ , and with lines 7 and 15 implemented with the EvalDotProduct, Algorithm 12. If no ciphertext shrinking is used, then the total number of NTTs is*

$$3 \cdot N \cdot d^2 \cdot \ell \cdot \left( \frac{N}{m^{\rho}} + \rho \cdot m \right). \quad (2)$$

*If we use shrinking at the end of each recursive call, then the total number of NTTs is*

$$\frac{3 \cdot N^2 \cdot d_{\rho}^2 \cdot \ell_{\rho}}{m^{\rho}} + 3 \cdot N \cdot m \cdot \left( \sum_{i=0}^{\rho-1} d_i^2 \cdot \ell_i \right) + 4 \cdot N \cdot \left( \sum_{i=0}^{\rho-1} d_i \cdot \ell_{i+1} \right) \quad (3)$$

where  $d_{\rho}$  and  $\ell_{\rho}$  define the dimension of the input ciphertexts (thus,  $d = d_{\rho} > d_{\rho-1} > \dots > d_0$  and  $\ell = \ell_{\rho} > \ell_{\rho-1} > \dots > \ell_0$ ).

*Proof.* The proof follows trivially by defining a recursive formula for the amount of NTTs per recursive call, then applying the number of NTTs already given in Algorithm 12. We provide a detailed proof in Appendix B.  $\square$

**Error growth of the homomorphic inverse NTT:** Assuming again that the sums in Algorithm 13 are implemented with the EvalScalarProd (Algorithm 12), we have the following result.

**Lemma 8.** *Consider the homomorphic evaluation of Algorithm 13 on input  $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{f_i}, E)$ , where  $0 \leq i < n$ . Let  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot (-s), E_s)$  and  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s(X^v), E_k)$  be the keys to compute the sums with Algorithm 12. Moreover, assume that  $E_s$  is constant. Then, the noise of the output ciphertexts is bounded by*

$$O\left((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^{\rho} \cdot \sqrt{N} \cdot (E + E_k)\right)$$

where  $\rho$  is the chosen recursive depth.



*Proof.* Algorithm 13 can be divided into three distinct steps:

- The splitting step, where the vectors  $\mathbf{g}$  passed to the recursive calls are defined. This does not introduce any additional error.
- The base case of the recursion, where the the NTT is computed with a quadratic algorithm.
- The recombination step, lines 12 to 15.

First consider the base case, where the quadratic step is applied to vectors of dimension  $N/m^\rho$ . In this step, each  $a_k$  is defined as the output of Algorithm 12 on ciphertexts  $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{f_i}, E)$ . Thus, by the noise analysis done in Algorithm 12, at the end of the base case, we have GSW ciphertexts with  $E^{(\rho)}$ -subgaussian noise, where

$$E^{(\rho)} = O \left( \left( \sqrt{\frac{N}{m^\rho}} \cdot \sqrt{d} \cdot p \cdot D \cdot \|s\| \right) \cdot E + \sqrt{dp} \cdot D \cdot E_s + \sqrt{\frac{N}{m^\rho}} \cdot \sqrt{p} \cdot \|s\| \cdot E_{KS} \right)$$

where  $E_{KS} = O(\sqrt{dp} \cdot D \cdot E_k)$ .

Now, for  $2 \leq i \leq \rho$ , let  $E^{(i)}$  be the subgaussian parameter of the noise of the ciphertexts used in the recombination step of the  $i$ -th recursive level. At this stage, each  $a_j$  is computed as an inner product with  $m$  coefficients, hence the noise in the output of the  $i$ -th recursive level is  $E^{(i-1)}$ -subgaussian, where  $E^{(i-1)}$  is obtained by applying the noise growth formula of Algorithm 12 with input noise  $E^{(i)}$  and vector dimension equal to  $m$ , that is,

$$E^{(i-1)} = O \left( \alpha \cdot E^{(i)} + \beta \right),$$

where  $\alpha = \sqrt{m \cdot d} \cdot D \cdot p \cdot \|s\|$  and  $\beta := \sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E_1 + \sqrt{p} \cdot \|s\| \cdot \sqrt{m} \cdot E_{KS}$ . Iterating this formula gives us

$$E^{(1)} = O(\alpha \cdot E^{(2)} + \beta) = \dots = O \left( \alpha^{\rho-1} \cdot E^{(\rho)} + \beta \sum_{i=0}^{\rho-2} \alpha^i \right).$$

By the definition of  $E^{(\rho)}$ , we have

$$E^{(1)} = O \left( \sqrt{N} \cdot (\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot E + \alpha^{\rho-1} \left( \sqrt{dp} \cdot D \cdot E_s + \sqrt{\frac{N}{m^\rho}} \cdot \sqrt{p} \cdot \|s\| \cdot E_{KS} \right) + \beta \sum_{i=0}^{\rho-2} \alpha^i \right).$$

Finally, by absorbing the lowest terms in the big-Oh notation, assuming that  $E_s$  is constant, and recalling that  $E_{KS} = O(\sqrt{dp} \cdot D \cdot E_k)$ , we obtain

$$\begin{aligned} E^{(1)} &= O \left( \sqrt{N} \cdot (\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot E + \alpha^{\rho-1} \left( \sqrt{\frac{N}{m^\rho}} \cdot \sqrt{d} \cdot D \cdot p \cdot \|s\| \cdot E_k \right) \right) \\ &= O \left( \sqrt{N} \cdot (\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot (E + E_k) \right) \end{aligned}$$

□

## 4.2 Partial decryption using $\text{NTT}_m^{-1}$

Remember that to decrypt  $(a, b) \in \hat{\mathcal{R}}_Q \text{LWE}_z(m, E)$ , we have to compute  $b^* := b - a \cdot z \bmod p$ , then it holds that  $b^* = e + \Delta \cdot m$ . Then we can recover each coefficient  $m_i$  by taking the  $\log p$  most significant bits. This step is cheap and easy to perform homomorphically. We call it message extraction and present it in Section 4.3. In this section, we present an algorithm that evaluates the main part of the decryption, that is, it uses the homomorphic inverse NTT to compute  $b^*$ .

The first part of the algorithm computes the forward NTT of  $a$  and of  $b$ , already scaled by the vector  $\psi$  of powers of a  $2N$ -root of unity, as described in Section 4.1. Then, given  $\bar{\mathbf{a}} := \text{NTT}(\psi \odot a)$  and the bootstrapping key encrypting  $\bar{\mathbf{z}} := \text{NTT}(-\psi \odot z)$ , we use GSW automorphisms to compute the entrywise product  $\bar{\mathbf{a}} \odot \bar{\mathbf{z}} = (\bar{a}_0 \cdot \bar{z}_0, \dots, \bar{a}_{N-1} \cdot \bar{z}_{N-1})$ . At this point, we also add  $\text{NTT}(\psi \odot b)$ . Finally, we apply the inverse NTT homomorphically to obtain GSW ciphertexts encrypting  $b^*$ .

---

### Algorithm 14: NTTDec, the homomorphic partial decryption

---

**Input:** Encryption  $\mathbf{c} \in \hat{\mathcal{R}}_p \text{LWE}_z(m, E^{(in)})$ . Bootstrapping keys  $\mathbf{K}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{-\bar{z}_i}, E)$ , where  $(\bar{z}_0, \dots, \bar{z}_{N-1}) := \text{NTT}(\psi \odot z) \in \mathbb{Z}_p^N$ , and key-switching keys for all the Galois automorphisms  $\eta_a : X \mapsto X^a$ . Vectors with powers of  $2N$ -th root of unity  $\psi$  in  $\mathbb{Z}_p$ , i.e.,  $\psi = (\psi^0, \dots, \psi^{N-1})$  and  $\psi^{-1} = (\psi^0, \dots, \psi^{-(N-1)})$

**Output:**  $\bar{\mathbf{C}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{e_i + \Delta \cdot m_i}, E'')$  for  $0 \leq i < N$

**Complexity:**  $O(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d^2 \cdot \ell)$  NTTs and  $O(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d \cdot \ell^2 \cdot p)$  products over  $\mathbb{Z}_{q_i}$

**Noise growth:**  $(E, E_k) \mapsto E'' = O((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{n \cdot p} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D))$

- 1 Parse  $\mathbf{c}$  as  $(a, b) \in \hat{\mathcal{R}}_p^2$  where  $\hat{\mathcal{R}}_p = \mathbb{Z}_p[X]/\langle X^N + 1 \rangle$
  - 2  $(\bar{a}_0, \dots, \bar{a}_{N-1}) \leftarrow \psi \odot \text{NTT}(a)$  ;  $\triangleright \text{NTT}(a) \in \mathbb{Z}_p^N$
  - 3  $(\bar{b}_0, \dots, \bar{b}_{N-1}) \leftarrow \psi \odot \text{NTT}(b)$  ;  $\triangleright \text{NTT}(b) \in \mathbb{Z}_p^N$
  - 4 **for**  $i \in \{1, \dots, n\}$  **do**
  - 5      $\bar{\mathbf{K}}_i = \eta_{\bar{a}_i}(\mathbf{K}_i)$  ;  $\triangleright \bar{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_{\eta_{\bar{a}_i}(s)}^d(1 \cdot X^{-\bar{a}_i \cdot \bar{z}_i}, E)$
  - 6      $\text{KS}_{\eta_{\bar{a}_i}(s) \rightarrow s}(\bar{\mathbf{K}}_i)$  ;  $\triangleright \bar{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{-\bar{a}_i \cdot \bar{z}_i}, E')$
  - 7      $\tilde{\mathbf{K}}_i = X^{\bar{b}_i} \cdot \bar{\mathbf{K}}_i$  ;  $\triangleright \tilde{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{\bar{b}_i - \bar{a}_i \cdot \bar{z}_i}, E')$
  - 8  $(\bar{\mathbf{C}}_0, \dots, \bar{\mathbf{C}}_{N-1}) = \text{NTT}^{-1}(\tilde{\mathbf{K}}_0, \dots, \tilde{\mathbf{K}}_{N-1})$  ;  $\triangleright \bar{\mathbf{C}}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{e_i + \Delta \cdot m_i}, E'')$
- $\triangleright E'' = O((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N \cdot p} \cdot (E' + E_k))$
- 

**Lemma 9 (Correctness, cost, and noise overhead of NTTDec).** *Given a ciphertext  $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_p \text{LWE}_z(\Delta \cdot m, E^{(in)})$ , the bootstrapping keys  $\mathbf{K}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot X^{\bar{z}_i}, E)$ , where  $(\bar{z}_1, \dots, \bar{z}_{N-1}) = \text{NTT}_p(\psi \odot -z)$ , and the keys used by Algorithm 12, namely  $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q \text{KS}_s^d(s(X^v), E_k)$  for all  $v \in \mathbb{Z}_p^*$  and  $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(1 \cdot (-s), E_s)$ , where  $E_s$  is a constant, then Algorithm 14 outputs GSW encryptions of  $X^{e_i + \Delta \cdot m_i}$  with  $E''$ -subgaussian noise, where*

$$E'' = O((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N \cdot p} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)).$$

Moreover, Algorithm 14 costs  $O(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d^2 \cdot \ell)$  NTTs and  $O(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d \cdot \ell^2 \cdot p)$  multiplication in  $\mathbb{Z}_{q_i}$ .

*Proof.* Let  $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_p \text{LWE}_z(m, E^{(in)})$ , then  $\bar{a}_i$  and  $\bar{b}_i$  are computed with a single NTT each, thus  $O(N \log N)$  operations over  $\mathbb{Z}_p$ , in clear. As these operations are not done homomorphically they do not contribute to the complexity, since the homomorphic operations dominate these.

In lines 5 and 6, we compute  $\bar{K}_i$  via GSW automorphism. Since this is done for each  $i$ , contributes a total of  $3 \cdot N \cdot d^2 \cdot \ell$  NTTs and  $O(N \cdot d \cdot \ell^2 \cdot p)$  modular multiplications.

Then, we perform  $N$  plaintext-ciphertext multiplications to add  $\bar{b}_i$  to the exponent. This costs zero NTTs and  $O(N \cdot p \cdot d \cdot \ell)$  modular multiplications, thus, it is already dominated by the cost of the automorphisms.

Finally, we apply the homomorphic inverse NTT, which, by Lemma 6, costs  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d^2 \cdot \ell\right)$  NTTs and  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d \cdot \ell^2 \cdot p\right)$  multiplications modulo  $q_i$ . Thus, it is clear that this step dominates the total cost.

As for the noise overhead, assuming that the ciphertexts defined in line 6 are obtained by applying the GSW Galois automorphisms defined in Algorithm 11, we have

$$E' = O(\sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E + \sqrt{dp} \cdot D \cdot E_k).$$

Then, multiplying by  $X^{\bar{b}_i}$  modulo  $X^p - 1$  only rotates the coefficients of the noise terms, but does not increase them. Thus, the ciphertexts  $\tilde{\mathbf{K}}_i$  also have  $E'$ -subgaussian noise.

Finally, by Lemma 8, the homomorphic inverse NTT increases the noise from  $E'$  to  $O\left((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N} \cdot (E' + E_k)\right)$ . Therefore, using the definition of  $E'$  and simplifying the expression by ignoring lower terms, we obtain

$$E'' = O\left((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N \cdot p} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right).$$

□

### 4.3 Message Extraction

After executing Algorithm 14, we obtain ciphertexts of the form  $\tilde{\mathbf{C}} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{\Delta \cdot m + e})$ , and we want to extract the message  $m$  from the exponent, moving it to the coefficient and deleting the noise term  $e$ , i.e., we want to map  $X^{\Delta \cdot m + e}$  to  $m$ . This message extraction procedure was introduced in [DM15] and adapted to different settings in subsequent works [CGGI16, Per21, BIP<sup>+</sup>22]. Usually, they assume we are working with negacyclic rings, i.e., modulo  $X^N + 1$ , and with binary messages, i.e.,  $m \in \{0, 1\}$ . Thus, in Algorithm 15, we provide a version of the message extraction algorithm adapted to the cyclic ring, assuming non binary messages, and also taking care of the scaling factor  $\alpha$  in the gadget matrix.

**Lemma 10 (Correctness of MsgExtract).** *Let the input ciphertext be  $\tilde{\mathbf{C}} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{\Delta \cdot m + e}, E)$ , with  $\|e\| < \Delta/2$ . Let the plaintext space be  $\mathbb{Z}_t$  for some  $t \geq 2$ . Let  $\mathbf{s} \in \mathbb{Z}^N$  be the coefficient vector of  $s$ . For any function  $f : \mathbb{Z}_t \rightarrow \mathbb{Z}_t$ , define the “test polynomial”  $t(X) \in \tilde{\mathcal{R}}$  as*

$$t(X) = X^{\Delta/2} \cdot \left( \sum_{i=0}^{\Delta-1} f(0) \cdot X^{p-i} + \sum_{i=\Delta}^{2\Delta-1} f(1) \cdot X^{p-i} + \dots + \sum_{i=(t-1) \cdot \Delta}^{t\Delta-1} f(t-1) \cdot X^{p-i} \right).$$

*Then, Algorithm 15, MsgExtract, outputs an LWE ciphertext  $\text{LWE}_s^p(\lfloor Q/t \rfloor \cdot f(m), E') \in \mathbb{Z}_Q^{p+1}$ , where  $E' = O(\sqrt{dp} \cdot D \cdot E)$ .*

---

**Algorithm 15: MsgExtract**


---

**Input:**  $\mathbf{C} \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{\Delta \cdot m+e}, E)$ , where  $\Delta = \lfloor p/t \rfloor$  and  $|e| < \Delta/2$ . A function  $f : \mathbb{Z}_t \rightarrow \mathbb{Z}_t$   
**Output:**  $\bar{\mathbf{c}} \in \text{LWE}_s^Q(\lfloor Q/t \rfloor \cdot f(m), E') \in \mathbb{Z}_t^{p+1}$   
**Complexity:**  $d \cdot \ell$  NTTs and  $O(\ell^2 \cdot p)$  products on  $\mathbb{Z}_{q_i}$ .  
**Noise growth:**  $E \mapsto O(\sqrt{dp} \cdot D \cdot E)$

- 1 Let  $t(X) = X^{\Delta/2} \cdot \sum_{i=0}^{t-1} \sum_{j=0}^{\Delta-1} f(i) X^{p-i\Delta-j} \bmod X^p - 1$
- 2 Let  $\bar{\mathbf{c}} = (0, \lfloor Q/t \rfloor \cdot t(X)) \in \tilde{\mathcal{R}}_Q^2$  be a trivial and noiseless encryption of  $t(X)$
- 3  $\mathbf{c}' := (a', b') = \mathbf{c} \boxtimes \mathbf{C}$  ;  $\triangleright \mathbf{c}' \in \tilde{\mathcal{R}}_Q \text{LWE}_s(\lfloor Q/t \rfloor \cdot t(X) \cdot X^{\Delta \cdot m+e}, E')$
- 4 Let  $\mathbf{A} \in \mathbb{Z}^{p \times p}$  be the circulant matrix of  $a'$
- 5 Let  $\mathbf{b} \in \mathbb{Z}^p$  be the coefficient vector of  $b'$
- 6 Let  $\mathbf{u} := (1, 0, \dots, 0) \in \{0, 1\}^p$
- 7 **return**  $\bar{\mathbf{c}} = [\mathbf{u} \cdot \mathbf{A}, \mathbf{u} \cdot \mathbf{b}]$  ;  $\triangleright \bar{\mathbf{c}} \in \text{LWE}_s^Q(\lfloor Q/t \rfloor \cdot f(m), E')$

---

*Proof.* The correctness follows from the standard observation the constant term of  $g(X) := t(X) \cdot X^{\Delta \cdot m+e}$  is equal to  $f(m)$ . This is essentially the same argument used in the extraction algorithms of [CGGI20, BIP<sup>+</sup>22]. Thus, by the correctness of the external product,  $\mathbf{c}'$  encrypts  $g(X)$  such that  $g_0 = f(m)$ .

The rest of the procedure just extracts from  $\mathbf{c}'$  an LWE sample corresponding to the first coefficient, hence, encrypting  $g_0$ .

The cost and noise growth are simply given by the external product, Algorithm 8, but with the number of NTTs divided by two, since the first polynomial in the external product is zero.  $\square$

#### 4.4 The Bootstrapping Algorithm

With all the sub-constructions in place, we can now fully define the bootstrapping algorithm and analyze both the complexity and the error growth.

The algorithm takes in  $N$  LWE samples  $\mathbf{c}_i \in \text{LWE}_s^Q(\Delta \cdot m_i) \in \mathbb{Z}^{p+1}$ , then packs them into one RLWE ciphertext  $\mathbf{c} \in \hat{\mathcal{R}}_Q \text{LWE}_z(\Delta \cdot m(X))$ , where  $m(X) = \sum m_i X^i$ . It proceeds by modulo switching  $\mathbf{c}$  from  $Q$  to  $p$  and running NTTDec, the partial decryption via homomorphic NTT, which generates GSW encryptions of  $X^{\Delta' m_i + e_i}$ . Finally, the messages  $m_i$  are extracted back into LWE ciphertexts. The bootstrapping is shown in detail in Algorithm 16.

---

**Algorithm 16: Bootstrap — for plaintext space  $\mathbb{Z}_t$** 


---

**Input:**  $\mathbf{c}_i \in \text{LWE}_s^Q(\Delta \cdot m_i, E^{(in)}) \in \mathbb{Z}^{p+1}$  for  $0 \leq i < N$ . All the bootstrapping and key-switching keys used in Algorithm 14. Any function  $f : \mathbb{Z}_t \rightarrow \mathbb{Z}_t$ .  
**Output:**  $\mathbf{c}'_i \in \text{LWE}_s^Q(\Delta \cdot f(m_i)) \in \mathbb{Z}^{p+1}$  for  $0 \leq i < N$   
**Complexity:** -  
**Noise growth:** -

- 1  $(a^{(1)}, b^{(1)}) = \text{PackLWE}(\mathbf{c}_0, \dots, \mathbf{c}_{N-1})$  ;  $\triangleright (a^{(1)}, b^{(1)}) \in \hat{\mathcal{R}}_Q \text{LWE}_z(\Delta \cdot m(X), E^{(1)})$
- 2  $(a^{(2)}, b^{(2)}) = \text{ModSwitch}_{Q \rightarrow p}(a^{(1)}, b^{(1)})$  ;  $\triangleright (a^{(2)}, b^{(2)}) \in \hat{\mathcal{R}}_p \text{LWE}_z(\Delta' \cdot m(X), E^{(2)})$
- 3  $(\mathbf{C}_0, \dots, \mathbf{C}_{N-1}) = \text{NTTDec}(a^{(2)}, b^{(2)})$  ;  $\triangleright \mathbf{C}_i \in \tilde{\mathcal{R}}_Q \text{GSW}_s^d(\alpha \cdot X^{e'_i + \Delta' \cdot m_i}, \bar{E})$
- 4 **for**  $0 \leq i < N$  **do**
- 5    $\mathbf{c}'_i = \text{MsgExtract}(\mathbf{C}_i)$  ;  $\triangleright \mathbf{c}'_i \in \text{LWE}_s^Q(f(m_i), E')$
- 6 **return**  $\mathbf{c}'_0, \dots, \mathbf{c}'_{N-1}$

---

**Lemma 11.** *Given at most  $N$  LWE ciphertexts and the keys described in Lemma 9, Algorithm 16 outputs LWE ciphertexts with  $E'$ -subgaussian noise, where*

$$E' = O\left((p \cdot \sqrt{d} \cdot D)^{\rho+1} \cdot \|s\|^\rho \cdot \sqrt{N} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right).$$

Moreover, it costs  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d^2 \cdot \ell\right)$  NTTs and  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho \cdot d \cdot \ell^2 \cdot p\right)$  multiplication in  $\mathbb{Z}_{q_i}$ .

*Proof.* The cost is asymptotically dominated by the NTTDec, thus, it follows directly from Lemma 9.

Again from Lemma 9, it holds that the noise of the GSW ciphertexts  $\mathbf{C}_i$  output by NTTDec satisfy  $\bar{E} = O\left((\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N} \cdot p \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right)$ , where  $E$  and  $E_k$  are the parameters of the noises from the bootstrapping keys and the key-switching keys, respectively. From Lemma 10, the final noise is then  $E' = O(\sqrt{dp} \cdot D \cdot \bar{E})$ . Hence, it holds that

$$\begin{aligned} E' &= O(\sqrt{dp} \cdot D \cdot (\sqrt{d} \cdot D \cdot p \cdot \|s\|)^\rho \cdot \sqrt{N} \cdot p \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)) \\ &= O\left((p \cdot \sqrt{d} \cdot D)^{\rho+1} \cdot \|s\|^\rho \cdot \sqrt{N} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right) \end{aligned}$$

□

**Corollary 2.** *The bootstrapping algorithm presented in Algorithm 16 has noise overhead of  $\tilde{O}(\lambda^{1.5+\rho})$ , where  $\lambda$  is the security parameter.*

*Proof.* For a security level of  $\lambda$  bits based on the RLWE problem, we can choose  $p, N \in \Theta(\lambda)$  and  $E, E_k, \|s\| \in O(1)$ . Since  $d = O(\log Q) = O(\log \lambda)$  and  $D$  is constant, we have

$$E' = O\left(\lambda^{\rho+1.5} \cdot (\log \lambda)^{(\rho+2)/2} \cdot D^{\rho+2} \cdot \|s\|^{\rho+1}\right) = \tilde{O}(\lambda^{1.5+\rho}).$$

□

**Theorem 1 (Correctness of bootstrapping).** *For a security parameter  $\lambda$ , let  $Q = \tilde{O}(\lambda^{2.5+\rho})$  and consider that the input ciphertexts  $\mathbf{c}_i \in \text{LWE}_s^Q(\Delta \cdot m_i, E^{(in)}) \in \mathbb{Z}^{p+1}$  satisfy  $E^{(in)} = O(Q/\lambda)$ . Then, with probability  $1 - 2^{-\lambda}$ , the output of Algorithm 16 is correct, that is, it outputs valid LWE encryptions of  $f(m_i)$ , with  $E'$ -subgaussian noise, where  $\tilde{O}(\lambda^{1.5+\rho}) = \tilde{O}(Q/\lambda)$ , thus, it is composable.*

*Proof.* From the description of the ring packing algorithm presented in Section 2.10, the parameter  $E^{(1)}$  shown in Algorithm 16 satisfy  $E^{(1)} = E^{(in)} \cdot \sqrt{N} + \sqrt{nN \log Q} \cdot E_R = O(E^{(in)} \cdot \sqrt{\lambda}) = O(Q/\sqrt{\lambda})$ , where we used  $n, N, \log Q \in O(\lambda)$  and  $E_R = O(1)$ .

Thus, after modulus switching, by using  $p = O(\lambda)$  and  $\|s\| = O(1)$ , it holds that  $E^{(2)} = O(E^{(1)}p/Q + \sqrt{N} \cdot \|s\|) = O(p/\sqrt{\lambda} + \sqrt{N} \cdot \|s\|) = O(p/\sqrt{\lambda})$ . Therefore, from the noise bound of the subgaussian distributions, as discussed in Section 2.3, the noise of the packed RLWE ciphertext is  $O(E^{(2)}\sqrt{\lambda}) = O(p)$  with probability  $1 - 2^{-\lambda}$ , which means it is decryptable. In other words, with overwhelming probability, the correctness condition of **MsgExtract** is satisfied, i.e., the restriction  $\|e\| < \Delta/2$  described in Lemma 10, since there  $\Delta = \lfloor p/t \rfloor = O(p)$ . Hence, the output indeed encrypts  $f(m_i)$ .

Moreover, by Corollary 2, the final noise is  $E'$ -subgaussian with  $E' = \tilde{O}(\lambda^{1.5+\rho})$ , which is  $O(Q/\lambda)$ , therefore, satisfies the same bound as the input, and thus, the bootstrapping is composable.

□

## 5 Practical Results

Amortized bootstrapping algorithms introduce significant asymptotic gains compared to non-amortized versions. However, they also introduce some performance overhead by requiring significantly larger parameters. As parameters grow, the asymptotic gains start to materialize, but, at the same time, memory requirements increase sharply to a point in which the implementation might become prohibitive. This is the problem preventing the [MS18] method from being practical, and, to a lesser extent, is also an issue in our method.

As such, our primary goal when developing a proof-of-concept practical implementation was to find the smallest parameter set in which our amortized bootstrapping starts to present practical gains. At first, we do not consider compression techniques in our search, as they generally would provide similar advantages and costs for all parameter sets. After defining our parameter sets, we discuss which compression techniques could be applied and present their gains in Section 5.4.

We implemented our scheme mostly from scratch using Intel HEXL Library [BKS<sup>+</sup>21] as the arithmetic backend to provide fast polynomial multiplications. We also adapted some basic functions from OpenFHE [BBB<sup>+</sup>22a] and MOSFHET [GBA22] libraries. We benchmarked our implementation in a `m5zn.metal` instance on AWS, featuring an Intel Xeon Platinum 8252C CPU at 4.5 GHz with 192 GB of RAM at 2933 MHz. We use Ubuntu 22.04 and G++ 11.3.0. Compiling options and further details are available in our repository.

### 5.1 Non-amortized version

We start with the implementation of the accumulator using the double-CRT CLWE variant of the GSW scheme described in Section 3. This scheme is as versatile as its cyclotomic counterpart, working with polynomials of any prime degree  $p$ . The main particularity is on how we implement multiplications modulo  $X^p - 1$ . Intel HEXL only provides fast polynomial multiplications based on the NTT for power-of-two cyclotomic rings  $\hat{\mathcal{R}} = \mathbb{Z}[X]/\langle X^{N'} + 1 \rangle$  while we need multiplications in  $\tilde{\mathcal{R}} = \mathbb{Z}[X]/\langle X^p - 1 \rangle$ . The most straightforward way of implementing them is to evaluate multiplications in  $\tilde{\mathcal{R}}$  for some  $N' > 2p - 2$ . In this way, the result has a degree at most  $(2p - 2)$  and is not reduced by the modulus  $X^{N'} + 1$ . This method presents two downsides. First, we need to execute the inverse NTT and manually evaluate the modular reduction  $X^p - 1$  after every multiplication. This is generally a minor issue, as, usually, these multiplications precede gadget decompositions, which would already require polynomials to be in coefficient representation. The second issue is that the size of our NTTs and the point-value representations of our polynomials is now  $N'$ , which is more than two times our RLWE dimension,  $p$ . This increases memory usage and slowdowns our basic operations proportionally. Optimally, we can choose values for  $p$  that are close to the next power of 2, minimizing the costs to still around two times. In Section 5.4, we discuss alternative ways of handling these issues that could be more efficient depending on parameters and the arithmetic backend.

As in a typical double-CRT implementation, we are free to choose the number of primes ( $\ell$ ) that compose the modulus and the number of digits ( $d$ ) we group them. We use 49-bit primes, based on the performance recommendations provided by HEXL [BKS<sup>+</sup>21]. Typically, we would use at least two primes per digit to reduce the execution time of gadget decompositions at the cost of a higher noise growth. However, the output noise of our amortized method relies heavily on the size of the digits ( $D$  in Lemma 8), thus requiring us to use just one prime per digit for all parameter sets that



we tested. Considering this, we decided to optimize our implementation for  $d = \ell$ , although we still provide all basic functions for  $d < \ell$ .

Table 2 shows the parameter sets and the execution times when evaluating the bootstrapping method defined in [LMK<sup>+</sup>22]. Their method is essentially the non-amortized version of our bootstrapping but defined for cyclotomic rings and small modulus. We cannot compare our implementation directly with theirs because they are using small parameters for bootstrapping 4-bit messages. While there is no intrinsic limitation to using double-CRT with small parameters, it is generally less efficient than working with binary decompositions. Double-CRT is broadly adopted in FHE for enabling larger precision and, thus, larger parameters. This is well exemplified in other cryptosystems [CKKS17,HPS19b]. In [LMK<sup>+</sup>22], if they wanted to bootstrap larger messages, they would also eventually need a larger modulus, which would also require double-CRT or some other technique for high-precision arithmetic.

**Table 2:** Execution time for the non-amortized bootstrapping. These result represent lower bounds, since they do not include the key switchings. The parameter  $n$  is the dimension of the input LWE ciphertexts.

#	$n$	$p$	$N'$	$\ell$	$d$	Execution time (ms)
1	512	16381	32768	2	2	2,300
2		32749	65536			5,613
3		12289	32768			2,328
4		15361				2,251
5	1024	12289				4,492

On the other hand, we can compare the cyclotomic and circulant versions of this bootstrapping, which we do in the first two rows of Table 2. Notice that the use of double-CRT is orthogonal to the choice of rings. We are measuring performance for our scheme bootstrapping in  $\tilde{\mathcal{R}}$ , but we are still computing in  $\hat{\mathcal{R}}$  in our arithmetic backend. Therefore, our results also approximately represent the performance of [LMK<sup>+</sup>22] over cyclotomic rings with dimension  $N'$  (for large parameters that require double-CRT). To be precise, the only differences between our implementation and a specialized one for  $\hat{\mathcal{R}}$  would be the absence of the modular reduction  $X^p - 1$  and the presence of tricks [LMK<sup>+</sup>22] to evaluate automorphisms with even generators. Both should be negligibly inexpensive and should not affect the comparison.

When implemented over the cyclotomic rings, [LMK<sup>+</sup>22] needs to deal with the negacyclicity, which reduces their bootstrapping capabilities. Specifically, it requires a padding bit at the beginning of the message unless they are using the bootstrapping to evaluate an anti-symmetric function, *i.e.*,  $f(x + N') = -f(x)$ . There are many different methods for avoiding negacyclicity and enabling a *full-domain evaluation* [KS21,CLOT21,GBA22], but, so far, all of them introduce slowdowns of at least 2 times and usually increase the output noise or probability of failure. Our scheme, on the other hand, naturally avoids negacyclicity. In this way, despite working in double the dimension with the same execution time, the cyclotomic version has essentially the same bootstrapping capabilities (assuming  $p \approx N'/2$ ) and, hence, the same performance level when considering the bootstrapping of arbitrary functions. Considering the particular case where the bootstrapping is used to evaluate anti-symmetric functions, the negacyclic version is indeed twice as fast, as we would need to use the parameter set 2 to match their negacyclic bootstrapping capabilities of parameter 1.

**Parameter selection, cleartext space, and bootstrapping capabilities** Although implementing and optimizing [LMK<sup>+</sup>22] to work with double-CRT (with and without CLWE) would be an interesting contribution per se, our goal with this implementation is to provide a comparison baseline for our amortized bootstrapping. As such, we choose parameters that enable similar bootstrapping capabilities as the ones chosen for the amortized version. In general, bootstrapping capabilities are solely defined by the three parameters: The output dimension  $p$ , the input error  $\sigma_{in}$ , and the output error  $\sigma_{out}$ . We consider and choose parameters such that  $\sigma_{out}$  is at least a few times smaller than  $\sigma_{in}$ , as one would reasonably expect from any bootstrapping procedure. In this way, the plaintext space can be defined based only on the first two parameters ( $p, \sigma_{in}$ ). Specifically, a  $k$ -bit message is correctly bootstrapped with probability  $\text{erf}\left(\frac{p/2^{k+1}}{\sigma_{in}\sqrt{2}}\right)$ , where  $\text{erf}$  is the Gauss error function and  $\sigma_{in}$  is the input error already mod switched to  $p$ . For some fixed dimension  $p$ , the plaintext space is then defined by  $\sigma_{in}$ , which, in turn, is defined by the input dimension  $n$  and, in a composed circuit (*i.e.*, when we bootstrap the result of a previous bootstrapping), the key switching errors introduced to change dimensions from  $p$  to  $n$ . From these, only the dimension  $n$  affects the bootstrapping, but it is mostly defined by how much noise the key switchings introduce and, thus, cannot be optimized without considering everything else.

Optimizing plaintext space is generally a complex task, requiring extensive parameter search, application considerations, and testing [BBB<sup>+</sup>22b]. In this work, instead of trying to find optimal parameters to maximize plaintext space, we evaluate the bootstrapping for two dimensions: 512 and 1024. The first is the smallest one that enables us to achieve the 128-bit security level with  $p \approx 2^{14}$  and a small error  $\sigma_{err} = 2$ . This is the worst-case scenario for our amortized bootstrapping, as it would enable higher gains in larger dimensions. Notice that, fixing all other parameters, the execution time increases linearly with the dimension in the non-amortized version but sublinearly in the amortized one (as our practical results show in Section 5.2). The second dimension is a much larger one, which enables a modulus of up to  $2^{25}$  bits, giving the key switching noise a lot of space to grow. The optimal solution for the dimension of the non-amortized version is some in between these two. However, as Section 5.2 also shows, our amortized bootstrapping is faster than the non-amortized one even when the first is operating in the largest dimension (1024) and the latter in the smallest (512). In this way, we show that the amortized version is faster regardless of what the optimal parameter is for the non-amortized one. Nonetheless, we also did measure the number of bits we were bootstrapping with our suboptimal parameters. For dimensions  $n = 512$  and  $n = 1024$ , respectively, we are able to bootstrap around  $4 \sim 5$  and  $7 \sim 8$  bits, with error rates smaller than 1% (calculated based on the measured standard deviation). We tested it both for the regular bootstrapping (evaluation of the rounding function) and arbitrary functions (functional bootstrapping [BGGJ19,CJP21]).

## 5.2 Amortized Bootstrapping

Compared to the non-amortize version, our amortized method introduces one additional restriction to the choice of parameters: There must exist a  $2N$ -th root of unity modulo  $p$ , where  $N$  is the dimension of the RLWE sample packing the input. Table 3 shows some of the possible values of  $p$  for  $N \in [512, 32768]$ . The minimum dimension for each value of  $p$  is defined by the security level of the input ring while the maximum is defined by the existence of a  $2N$ -th root of unity modulo  $p$ . We do not present results for  $N < 512$  because it would be necessary to adapt our scheme to module-RLWE to achieve the 128-bit security level in these small dimensions. For all parameters,

we use secret keys sampled from a Gaussian distribution with  $\sigma_s = 3.2$  and adjust the noise to the minimum necessary to achieve at least the 128-bit security level. We refer to Appendix ?? for a summary of the parameters used in the bootstrapping.

**Table 3:** List of prime moduli and respective possible input dimensions for the amortized bootstrapping.

$p$	$\log_2(Q)$	$N$	
		min.	max
12289	13.59	512	2048
15361	13.91	512	512
40961	15.32	1024	4096
65537	16.00	1024	32768

**INTT Performance** The homomorphic evaluation of the INTT (Algorithm 13) is the core and most expensive procedure in our amortized bootstrapping. Table 4 shows its execution time for  $\ell = d = 4$  and  $\rho = 2$ , with and without shrinking.

**Table 4:** Execution time, in milliseconds, for the INTT for  $\ell = 4$  and  $\rho = 2$ .

$p$	$N$	Without shrinking		With shrinking		Shrinking Speedup
		Exec. Time	Amortized Time	Exec. Time	Amortized Time	
12289	512	1,639,917	3,203	1,079,466	2,108	1.52
12289	1024	5,416,006	5,289	3,182,273	3,108	1.70

*Remark 1 (INTT maximum depth).* Our INTT (Algorithm 13) runs in time  $O\left(N^{1+\frac{1}{\rho}} \cdot \rho\right)$ . Increasing the maximum depth  $\rho$  from 1 (quadratic) to 2 results in a 11 times gain for  $N = 512$  and 16 times for  $N = 1024$ . From 2 to 3, gains for both dimensions are just around  $1.8 \sim 2.1$  times. After that, they only continue to decrease. On the other hand, every time we increase  $\rho$ , we multiply the output error by at least  $\sqrt{d}pD \|s\|$  (Lemma 8). The greatest factor in this equation is  $D$ , which, by itself, requires us to use  $\frac{\ell}{d}$  more primes ( $\ell$ ) every time  $\rho$  increases. Performance, in turn, degrades cubically in  $\ell$ , which starts in 2 (non-amortized version). Increasing  $\rho$  from 1 to 2 increases  $\ell$  from 2 to 3, introducing a slowdown of 3.4 times. This is significantly below the 11 (or 16 for  $N = 1024$ ) times speedup this change enables in the INTT. However, increasing  $\rho$  from 2 to 3 enables gains of up to 2.1 times while increasing  $\ell$  from 3 to 4 slows down the implementation 2.4 times. This estimate does not consider hidden constants, but we verified it experimentally for our parameters and concluded  $\rho = 2$  is the optimal value in our case.

**Bootstrapping Performance** As we move to the complete bootstrapping, we introduce the overhead of calculating several key switchings, but we also can further optimize the INTT evaluation. As we defined, it outputs GSW ciphertexts, from which we extract an RLWE and continue to the message extraction phase of the bootstrapping. A more efficient way of implementing it is to run

the last recombination step (Lines 12 to 15 in Algorithm 13) already using an RLWE as the accumulator for the summations. Further, we can also initialize the accumulator with the test vector at the beginning of this last recombination (similarly as introduced in [CGGI16]), which simplifies the message extraction and reduces the output error. This also allows us to use a smaller modulus, with  $\ell = 3$ . Table 5 shows the results for complete bootstrapping, including key switchings, with  $\rho = 2$ .

**Table 5:** Execution time, in milliseconds, for the amortized bootstrapping. Speedup is over the fastest parameter of the non-amortized bootstrapping.

$p$	$N$	$\ell = d$	Total Time	Amortized Time	Speedup
12289	512	3	435,024	850	2.71
15361	512	3	445,950	871	2.64
12289	512	4	888,887	1,736	1.32
12289	1024	3	1,415,236	1,382	1.66
12289	1024	4	2,939,374	2,870	0.80

*Remark 2 (Bootstrapping with GSW output).* We note, however, that this is not a general optimization, since having a bootstrapping with GSW output is also useful for some applications. One example is in the context of the Functional Bootstrapping [BGGJ19,CJP21], a technique that uses the bootstrapping to evaluate arbitrary functions represented as lookup tables (LUT) encoded in the test vector. In this context, the GSW output can be used to evaluate private functions (encrypted LUTs), evaluate multivariate functions, and perform the multi-digit evaluation of very large LUTs [GBA21]. In this case, we refer to the execution times we provided in Table 4, but we also note these techniques may require different types of key switchings, which would also introduce some additional computation overhead.

The high memory requirements are certainly the main drawback of our construction, but there already are several ways of mitigating it. Table 6 shows the detailed memory requirements for our parameters. Compressed (Compr.) values consider different techniques that we can use to reduce memory requirements, but that might affect performance. Performance Optimal (Perf. Opt.) values do not consider any compression techniques, focusing solely on improving performance. Our implementation and, hence, all execution times we present in this section do not consider the compression techniques. LWE and Packing key switching keys are not included in this comparison. The table considers the following techniques:

- Temporary buffer. Our algorithm needs a temporary buffer because it does not calculate the INTT in place. Instead of having a different buffer for each call of  $trivial - NTT^{-1}$ , we can have a single one shared among them. This restricts parallelization and does not allow memory from shrunk ciphertexts to be freed.
- State buffer. This is the array of accumulators. To save memory, we can evaluate the INTT directly over the bootstrapping key. This comes at the cost of having to load the bootstrapping key before each bootstrapping.
- Automorphism key switching keys. We can use decomposed automorphism calculations, as we further discuss in Section 5.4.

**Table 6:** Memory requirements for  $p = 12289$ .

	$N = 512$				$N = 1024$			
	$\ell = 3$		$\ell = 4$		$\ell = 3$		$\ell = 4$	
	Perf. Opt.	Compr.	Perf. Opt.	Compr.	Perf. Opt.	Compr.	Perf. Opt.	Compr.
Bootstrapping Key	4.5 GiB	4.5 GiB	8.0 GiB	8.0 GiB	9.0GiB	9.0 GiB	16.0 GiB	16.0 GiB
Automorphism key	54.0 GiB	63.0 MiB	96.0 GiB	112.0 MiB	54.0GiB	63.0 MiB	96.0 GiB	112.0MiB
Temporary buffer	2.2 GiB	72.0 MiB	4.0 GiB	128.0 MiB	4.5GiB	72.0 MiB	8.0 GiB	128.0MiB
State buffer	4.5 GiB	0	8.0 GiB	0	9.0GiB	0	16.0 GiB	0
Total	65.3 GiB	4.6 GiB	116.0GiB	8.2 GiB	76.5GiB	9.1 GiB	136.0 GiB	16.2 GiB

### 5.3 Packing Key Switching

Our bootstrapping capabilities are significantly limited by the amount of error in the RLWE sample packing our input. As such, it is essential to reduce the error added by the Packing Key Switching procedure as much as we can. Following the original blueprint of [MS18], we would reduce the dimension of the output LWE samples at the end of our bootstrapping procedure to the dimension of our input RLWE. Then, we would just pack these samples at the beginning of a new bootstrapping. In this way, packing occurs directly in the input dimension. This works well for  $n = 1024$ , as we can have a large modulus accommodating the packing error. However, with  $n = 512$ , the modulus is already  $p$ , which supports significantly less noise. In this work, we propose a different strategy for significantly reducing noise growth in the packing procedure.

1. At the end of the amortized bootstrapping, we reduce the dimension of each LWE sample to an intermediary one. For our practical implementation with  $n = 512$ , we choose it to be  $2n = 1024$ . We also mod switch to an intermediary modulus chosen according to the security level, in our case  $p^* \approx 2^{25}$ .
2. At the beginning of the bootstrapping, we pack the LWE samples still in the intermediary dimension to an RLWE of the same dimension. At this point, we have two options, we can pack it an RLWE of dimension  $N = 1024$ , or to module-RLWE sample with  $k = 2$  and  $N = 512$ , where  $k$  is the number of polynomials in the vector  $a$  of the module-RLWE scheme. For the practical implementation, we choose the latter.
3. We reduce the dimension of the RLWE sample using an RLWE key switching.

### 5.4 Further Improvements

Our implementation is a proof of concept to show that efficient amortized bootstrappings are practical. As such, we present only the optimizations necessary for achieving this goal. However, since our method is based on several well-studied and broadly used constructions, many other promising techniques could be applied to further improve performance. In this section, we briefly discuss a few of them, leaving their actual implementation as future work. We note that, although we estimate the impact for some of them using experimental data, we do not present these techniques in our implementation.

**Optimal Parameter Selection** We choose parameters in Section 5.1 based on a manual search of parameters. As we discussed there, a formal parameter search method is necessary to optimize the size of messages we can bootstrap. The same goes for the key switchings described in Section 5.3.

Particularly, for our current results, the sequences of key switchings that precede the bootstrapping are the limiting factor for the message size, as we are obtaining similar error rates for  $\ell = 3$  and  $\ell = 4$ .

**Decomposed Automorphisms** In [LMK<sup>+</sup>22], they introduce an efficient way of reducing the number of automorphism keys by representing the automorphism generators as a product of the powers of the ring multiplicative generator. In their case, automorphism generators are in  $\mathbb{Z}_{2N}^*$ , and they need to map them to  $\mathbb{Z}_{N/2} \times \mathbb{Z}_2$  to obtain a multiplicative generator. In ours, automorphism generators are in  $\mathbb{Z}_p^*$  for some prime  $p$ , so we always have a multiplicative generator of order  $p - 1$ . This technique can reduce the number of automorphism key switching from  $p$  down to just  $\lceil \log_2(p) \rceil$ , which would reduce the total size of the automorphism keys 877 times for our parameters. In terms of performance, we estimate a slowdown of around 2 to 3 times. In [LMK<sup>+</sup>22], they also introduce the concept of *window size*, which defines the number of automorphism keys they actually use. In this way, they are able to provide intermediary solutions, which require larger keys but introduce less performance overhead.

**Reducing the size of NTTs** When defining the accumulator for the non-amortized version, we are able to select values for  $p$  that are very close to the next power of two. The same cannot be done when defining  $p$  for the amortized method, as we are further limited by the necessity of a  $2n$ -th root of unity modulo  $p$ . If the goal is just to reduce memory usage, we can use the Bluestein NTT [Blu70], which allows evaluating multiplications directly in  $\tilde{\mathcal{R}}$ . However, Bluestein essentially turns the transform in a convolution of size  $2N$ , thus not improving itself. In fact, it might introduce additional overheads to the transform computation. It also requires the existence of  $p$ -th roots of unity modulo the primes  $Q_i$ , which further limits our parameter selection. Nonetheless, it would reduce our memory requirements by around two times.

## Acknowledgments

This work has been supported in part by Cyber Security Research Flanders with reference number VR20192203, by the Defence Advanced Research Projects Agency (DARPA) under contract No. HR0011-21-C-0034 DARPA DPRIVE BASALISC, and by the FWO under an Odysseus project GOH9718N.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, the US Government, or Cyber Security Research Flanders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

This work was done while A. Guimarães was visiting the Department of Computer Science at Aarhus University. He is supported by the São Paulo Research Foundation under grants 2013/08293-7, 2019/12783-6, and 2021/09849-5.

## References

- AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.



- BBB<sup>+</sup>22a. Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022. <https://eprint.iacr.org/2022/915>.
- BBB<sup>+</sup>22b. Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization & larger precision for (T)FHE. Cryptology ePrint Archive, Report 2022/704, 2022. <https://eprint.iacr.org/2022/704>.
- BDF18. Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*, volume 10831 of *Lecture Notes in Computer Science*, pages 217–251, Marrakesh, Morocco, May 7–9, 2018. Springer, Heidelberg, Germany.
- BGGJ19. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Simulating homomorphic evaluation of deep learning predictions. Cryptology ePrint Archive, Report 2019/591, 2019. <https://eprint.iacr.org/2019/591>.
- BGH13. Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 1–13, Nara, Japan, February 26 – March 1, 2013. Springer, Heidelberg, Germany.
- BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.
- BIP<sup>+</sup>22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. Final: Faster fhe instantiated with ntru and lwe. In *Advances in Cryptology – ASIACRYPT 2022*, Cham, 2022. Springer International Publishing.
- BKS<sup>+</sup>21. Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D.M. de Souza, and Vinodh Gopal. Intel HEXL: Accelerating homomorphic encryption with intel AVX512-IFMA52. Cryptology ePrint Archive, Report 2021/420, 2021. <https://eprint.iacr.org/2021/420>.
- Blu70. L. Bluestein. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.
- CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.
- CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
- CJP21. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Report 2021/091, 2021. <https://eprint.iacr.org/2021/091>.
- CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.
- CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 670–699, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.
- DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
- FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. <https://eprint.iacr.org/2012/144>.
- GBA21. Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, 2021. <https://tches.iacr.org/index.php/TCHES/article/view/8793>.

- GBA22. Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized software for FHE over the torus. Cryptology ePrint Archive, Report 2022/515, 2022. <https://eprint.iacr.org/2022/515>.
- Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.
- GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.
- GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.
- HPS19a. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 83–105, San Francisco, CA, USA, March 4–8, 2019. Springer, Heidelberg, Germany.
- HPS19b. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, Lecture Notes in Computer Science, pages 83–105, Cham, 2019. Springer International Publishing.
- HS15. Shai Halevi and Victor Shoup. Bootstrapping for HELib. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
- KPZ21. Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 608–639, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.
- KS21. Kamil Klucznik and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/1135, 2021. <https://eprint.iacr.org/2021/1135>.
- LMK<sup>+</sup>22. 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. Cryptology ePrint Archive, Report 2022/198, 2022. <https://eprint.iacr.org/2022/198>.
- LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.
- MS18. Daniele Micciancio and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamani, Dániel Marx, and Donald Sannella, editors, *ICALP 2018: 45th International Colloquium on Automata, Languages and Programming*, volume 107 of *LIPIcs*, pages 100:1–100:14, Prague, Czech Republic, July 9–13, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.
- Nus80. H. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, 1980.
- Per21. Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 331–359, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany.
- Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

## A Generalization of circulant-LWE encryption

In [BDF18], it was proved that we can use the circulant-RLWE problem to encrypt messages of the form  $X^k$ , that is, powers of  $X$ . We now show that by slightly modifying their proof, we can



prove that a scheme that encrypts polynomials  $m(X)$  whose sum of coefficients is equal to 0 is also CPA-secure under the RLWE assumption.

**Lemma 12.** *If the decisional Ring-LWE problem is hard for the prime-order cyclotomic polynomial  $\Phi_p(X)$ , a modulus  $Q \in \mathbb{Z}$ , and standard deviation  $\sigma$ , then the Circulant-LWE scheme is cpa-secure for messages of the form  $m(X) = \sum_{i=0}^{p-1} m_i \cdot X^i$  where  $m(1) = 0$ .*

*Proof.* Let  $S_{Q,p} := \{\sum_{i=0}^{p-1} a_i X^i : \sum_{i=0}^{p-1} a_i = 0 \bmod Q\}$ . By Lemma 11 of [BDF18], if the Ring-LWE problem is hard, then the Circulant-LWE distribution is indistinguishable from the uniform distribution over  $S_{Q,p}^2$ . Now, notice that for any  $\Delta \in \mathbb{Z}$  and  $b \in S_{Q,p}$ , if we define  $c = b + \Delta \cdot m$ , then we have

$$\sum_{i=1}^{p-1} c_i = \sum_{i=1}^{p-1} b_i + \Delta \sum_{i=1}^{p-1} m_i = 0 \bmod Q.$$

Thus,  $S_{Q,p} + \Delta m = S_{Q,p}$ . Therefore, the circulant-LWE encryption of  $m$  is indistinguishable from a uniformly random sample from  $S_{Q,p} \times (S_{Q,p} + \Delta m) = S_{Q,p}^2$ .  $\square$

Now, notice that the secret key of the Circulant-LWE scheme is obtained by projecting the secret  $\bar{s}$  from the RLWE to  $s = L((1 - X)\bar{s}) \in S_{Q,p}$ , therefore, it satisfies  $s(1) = 0$ . Moreover, applying automorphisms to  $s$  modulo  $X^p - 1$  only reorder the coefficients, so, if  $s^{(k)}(X) := s(X^k)$ , we also have  $s^{(k)}(1) = 0$ , therefore, encrypting  $s^{(k)}(X)$  for any  $k$  is secure and we conclude that it is safe to publish the key-switching keys from  $s$  to  $s^{(k)}$ . Finally, in Algorithm 11, we also need to encrypt a product  $s \cdot z$ , but both keys are also CLWE secrets, thus, we still have  $s(1) \cdot z(1) = 0$ .

## B Number of NTTs in homomorphic inverse NTT

Let  $T(N)$  be the number of NTTs over  $\mathbb{Z}_{q_i}$  that are executed during the homomorphic evaluation of the inverse NTT, Algorithm 13. Firstly, let's consider that no ciphertext shrinking is used, so  $d$  and  $\ell$  are constant in all the recursive levels. We can see that  $T(N) = m \cdot T(N/m) + R(N)$  where  $m \cdot T(N/m)$  accounts for the  $m$  recursive calls on vectors of dimension  $N/m$  and  $R(N)$  is the number of NTTs of the recombination step. Assuming that the sums are implemented with EvalScalarProd, Algorithm 12, which cost  $3 \cdot m \cdot d^2 \cdot \ell$  NTTs, we have  $R(N) = N \cdot (3 \cdot m \cdot d^2 \cdot \ell) = 3 \cdot N \cdot m \cdot d^2 \cdot \ell$ . Therefore,

$$T(N) = m \cdot T(N/m) + 3 \cdot N \cdot m \cdot d^2 \cdot \ell.$$

Iterating this up to  $\rho$  recursive levels, we have

$$\begin{aligned} T(N) &= m \cdot T(N/m) + 3 \cdot N \cdot m \cdot d^2 \cdot \ell \\ &= m \cdot (m \cdot T(N/m^2) + 3 \cdot (N/m) \cdot m \cdot d^2 \cdot \ell) + 3 \cdot N \cdot m \cdot d^2 \cdot \ell \\ &= m^2 \cdot T(N/m^2) + 3 \cdot 2 \cdot N \cdot m \cdot d^2 \cdot \ell \\ &= \vdots \\ &= m^\rho \cdot T(N/m^\rho) + 3 \cdot \rho \cdot N \cdot m \cdot d^2 \cdot \ell \end{aligned}$$

Then, we run the homomorphic quadratic inverse NTT on dimension  $N/m^\rho$ . Since on dimension  $k$ , this algorithm executes  $k$  times the homomorphic scalar product of dimension  $k$ , its cost is

$k \cdot (3 \cdot k \cdot d^2 \cdot \ell)$ . Therefore, using  $k = N/m^\rho$ , the total number of NTTs is

$$T(N) = m^\rho \cdot 3 \cdot (N/m^\rho)^2 \cdot d^2 \cdot \ell + 3 \cdot \rho \cdot N \cdot m \cdot d^2 \cdot \ell = 3 \cdot N \cdot d^2 \cdot \ell \cdot \left( \frac{N}{m^\rho} + \rho \cdot m \right).$$

To generalize this analysis to the case where shrinking is used, let's assume that we shrink the ciphertexts at the end of each recursive call. Thus, considering that we have recursive depth  $\rho$ , we have different dimensions and number of primes,  $d_i$  and  $\ell_i$  for  $0 \leq i \leq \rho$ , where  $\ell_\rho$  is the number of primes that we have in the very beginning (thus,  $d_\rho > d_{\rho-1} > \dots > d_0$  and  $\ell_\rho > \ell_{\rho-1} > \dots > \ell_0$ ).

Now, the formula for the number of NTTs becomes

$$T(N, i) = m \cdot T(N/m, i+1) + 3 \cdot N \cdot m \cdot d_i^2 \cdot \ell_i + 4 \cdot N \cdot d_i \cdot \ell_{i+1}$$

where the last term,  $4 \cdot N \cdot d_i \cdot \ell_{i+1}$ , accounts for the number of NTTs executed by the shrinking algorithm to switch from dimension  $(d_{i+1}, \ell_{i+1})$  to  $(d_i, \ell_i)$  the  $n$  ciphertexts output by the  $m$  recursive calls.

By starting with  $i = 0$  and iterating  $\rho$  times again, we have

$$\begin{aligned} T(N, 0) &= m \cdot T(N/m, 1) + 3 \cdot N \cdot m \cdot d_0^2 \cdot \ell_0 + 4 \cdot N \cdot d_0 \cdot \ell_1 \\ &= m^2 \cdot T(N/m^2, 2) + 3 \cdot N \cdot m \cdot d_1^2 \cdot \ell_1 + 4 \cdot N \cdot d_1 \cdot \ell_2 + 3 \cdot N \cdot m \cdot d_0^2 \cdot \ell_0 + 4 \cdot N \cdot d_0 \cdot \ell_1 \\ &= m^2 \cdot T(N/m^2, 2) + 3 \cdot N \cdot m \cdot (d_1^2 \cdot \ell_1 + d_0^2 \cdot \ell_0) + 4 \cdot N \cdot (d_1 \cdot \ell_2 + d_0 \cdot \ell_1) \\ &= \vdots \\ &= m^\rho \cdot T(N/m^\rho, \rho) + 3 \cdot N \cdot m \cdot \left( \sum_{i=0}^{\rho-1} d_i^2 \cdot \ell_i \right) + 4 \cdot N \cdot \left( \sum_{i=0}^{\rho-1} d_i \cdot \ell_{i+1} \right) \end{aligned}$$

Now, using  $3 \cdot (N/m^\rho)^2 \cdot d_\rho^2 \cdot \ell_\rho$  for the number of NTTs of the quadratic step, i.e.,  $T(N/m^\rho, \rho)$ , the total number of NTTs used in the homomorphic inverse bootstrapping with one layer of shrinking after each recursive level is

$$T(N, 0) = \frac{3 \cdot N^2 \cdot d_\rho^2 \cdot \ell_\rho}{m^\rho} + 3 \cdot N \cdot m \cdot \left( \sum_{i=0}^{\rho-1} d_i^2 \cdot \ell_i \right) + 4 \cdot N \cdot \left( \sum_{i=0}^{\rho-1} d_i \cdot \ell_{i+1} \right)$$

## C Glossary of parameters used in this work

Parameter	Description	Size
$\lambda$	Security level	—
$n$	Dimension of LWE samples	$O(\lambda)$
$N$	Degree of RLWE samples $(X^N + 1)$ . Power of two.	$O(\lambda)$
$p$	Degree of CLWE samples $(X^p - 1)$ . Prime.	$O(\lambda)$
$\rho$	Recursive depth of the bootstrapping algorithm	—
$q_i$	Prime factor of ciphertext modulus $Q$	$< 2^{50}$
$Q$	Ciphertext modulus defined as $\prod_{i=1}^{\ell} q_i$	$\tilde{O}(\lambda^{1.5+\rho})$
$\ell$	Number of prime factors of $Q$	$O(\rho \cdot \log \lambda)$
$t$	Plaintext modulus	$O(1)$
$d$	Dimension of GSW ciphertexts. Number of CRT digits	$O(\ell)$
$D_i$	Each CRT digit. Product of approx. $\ell/d$ primes.	$O(q_i \cdot \ell/d)$
$D$	Upper-bound on the size of the CRT digits	$\max(D_1, \dots, D_d)$