Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented

Antonio Guimarães[1], Hilder V. L. Pereira[1], and Barry van Leeuwen[2]

[1] Institute of Computing – University of Campinas, Brazil⋆⋆⋆,
[2] COSIC, KU Leuven, Leuven, Belgium.†

antonio.guimaraes@ic.unicamp.br,
hilder@ic.unicamp.br,
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 could ever be practical, which reduces the impact of their results. In this work, we follow their general framework, but propose an amortized bootstrapping procedure that is conceptually simpler and asymptotically cheaper. We reduce the number of homomorphic multiplications 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 $\widetilde{O}(n^{2+3\cdot\rho})$ to $\widetilde{O}(n^{1+\rho})$, where n is the security level and $\rho \geq 1$ is a free parameter. We also make it more general, by handling non-binary messages and applying programmable bootstrapping. To obtain a concrete instantiation of our bootstrapping algorithm, we describe 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 also provide a C++ implementation of our algorithm, thus showing for the first time the practicability of the amortized bootstrapping. Moreover, it is competitive with existing bootstrapping algorithms, being even around 3.4 times faster than an equivalent non-amortized version of our bootstrapping.

1 Introduction

Since the introduction of the first Fully Homomorphic Encryption (FHE) scheme, by Gentry [16], 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* operation that refreshes the ciphertexts after being involved in a few homomorphic operations, allowing us to perform further operations on them. Hence, most works aiming to make FHE more efficient direct their efforts towards designing faster bootstrapping. For this goal, there are two main strategies:

Heavy-packed bootstrapping tries to pack several messages into a "large" ciphertext. This makes bootstrapping complex and very costly, but refreshes several messages at once, with the aim for the *amortized* cost per message to be low. This type of bootstrapping was proposed for many schemes, [11,15,10,17], obtaining good amortized costs, however these schemes generally were not very efficient regarding noise management, thus, their bootstrapping algorithms often incur quasi-polynomial noise growth. This implies that their security is based on worst-case lattice problems with *superpolynomial approximation factors*. Ideally, we would like to have FHE with assumptions identical to general lattice-based public-key encryption, which assumes worst-case lattice problems with *polynomial approximation factors* only.

⋆⋆⋆ This paper was mainly written while Hilder V. L. Pereira was in COSIC, KU Leuven.

† Please, notice that *van Leeuwen* is the family name of the third author, thus, instead of [GPL23], the correct acronym for citing this paper would be [GPV23] or maybe [GPvL23].

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.*

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

On the other front, fast single message bootstrapping encrypts a single message into a "small" ciphertext, hugely simplifying the bootstrapping. The aim here is to execute it much faster, in many cases in a few milliseconds on a common commercial computer [14,12,32,8]. The downside now is the need for one bootstrapping per gate of the circuit being evaluated homomorphically and, since each bootstrapping refreshes a single message, the amortized cost is still high. However, this bootstrapping strategy does attain a polynomial noise overhead, achieving the ideal assumption base: worst-case lattice problems with polynomial approximation factors.

Then, in [29], Micciancio and Sorrell try to obtain the advantages of these two approaches, by proposing a bootstrapping algorithm that follows the blueprint of [14], 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. Due to the limitations of the functions one can evaluate homomorphically, they cannot simply evaluate a Fast Fourier Transform, thus, they adapt the Nussbaumer Transform [31] 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 ρ is a free parameter. Note that the $\tilde{O}$-notation 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 [29], 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 make the whole bootstrapping algorithm more straightforward and 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 $\tilde{O}(n^{1+\rho})$. Moreover, instead of just bits, we can handle n messages in $\mathbb{Z}_t$, for small t. This also means that we support programmable bootstrapping, reducing the noise and simultaneously applying any function $f : \mathbb{Z}_t \to \mathbb{Z}_t$ to the message. In Table 1, we present a comparison of our work with previous ones.

Although [29] 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 formalize a double-CRT (RNS) variant of the GSW scheme [18], 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. 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 of times that the NTT is executed is already used to estimate the cost of several previous schemes, such as [14,12,8].

Finally, we also implemented our bootstrapping in C++ and made it publicly available,[3] thus, presenting the first implementation of amortized and providing baseline running times and memory usage for this type of bootstrapping, showing that such a scheme is feasible in practice with running times comparable to some existing schemes.

1.1 Overview of the amortized bootstrapping from [29]

The bootstrapping strategy of [3], improved and made practical in [14], works as follows: the whole FHE scheme is organized in two layers, each one composed by one homomorphic scheme. The base scheme is an LWE-based scheme that can perform very limited number of homomorphic operations, then has to be bootstrapped. Then the second scheme, called the accumulator, is used to evaluate the decryption of the base scheme homomorphically, i.e., to bootstrap it. For the accumulator, one uses the GSW scheme [18] 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 is just polynomial in the security parameter. 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 [29] is to combine $O(n)$ LWE ciphertexts into one single RLWE ciphertext $\mathbf{c} \in \mathcal{R}^2$ encrypting $O(n)$ messages. Then, because 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 one 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, [29] sets the recursion level as a parameter ρ.

Moreover, since the GSW scheme is instantiated over the ring $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$ and working only 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, [29] 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., they obtain r inputs of length $3n/r$ instead of length n/r. Since this expansion happens in all recursive levels, the factor 3 accumulates exponentially and, 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 $\tilde{O}(n^{2+3\cdot\rho})$.

[3] GitHub repository: https://github.com/antoniocgj/Amortized-Bootstrapping

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 vice versa. As the Number Theoretic Transform (NTT) is the algorithm of choice to perform multiplications modulo $X^N + 1$ our goal is to obtain an accumulator that can evaluate NTTs.

To obtain that, we use the results from [7] 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 an equivalent instruction set of [29], 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 [7,24] to the GSW scheme, which expands the instruction set of our accumulator. In Table 2, we compare both accumulators. Putting it all together we obtain a GSW-based accumulator that allows us to homomorphically evaluate a standard NTT. The only limitation that remains is that the noise overhead of the bootstrapping is still exponential in the number of recursive levels of the NTT, hence restricting to ρ recursive levels as in [29], guarantees that the noise overhead remains 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^ρ in the time complexity and to reduce the noise overhead from $\tilde{O}(n^{2+3\cdot\rho})$ to $\tilde{O}(n^{1+\rho})$.

Additionally, our accumulator also allows us to replace the algorithm used in [29] 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 [29] to $O(\rho \cdot n^{1+1/\rho})$. In Figure 1, we present the main steps of our bootstrapping.

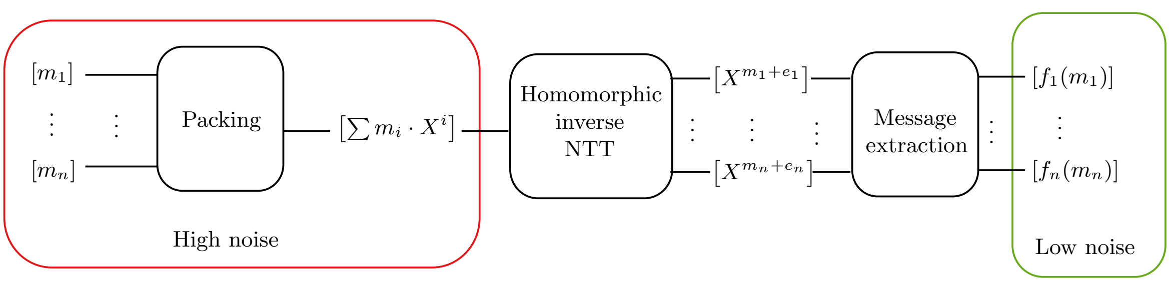

Fig. 1: *Main building blocks of our amortized bootstrapping. First we pack high-noise ciphertexts into a single ciphertext encrypting a polynomial with original messages as the coefficients. Then we evaluate the NTT homomorphically, obtaining encryptions of powers of X having the messages plus restricted noise in the exponent. Finally, we execute a message extraction procedure, removing the noise terms and applying any set of desired functions to the messages.*

Table 2: *Comparison of the accumulator proposed in [29] and ours. The notation $[a]$ means encryption of (X to the power of) a. Negation is not natively supported by [29], thus, for any message m, they actually encrypt $(m, -m)$, which requires two ciphertexts. Negation is then implemented by swapping the ciphertexts so that they encrypt $(-m, m)$. However this doubles the memory and time of all their operations.*

	Variable type	Size of enc. message	Available operations				
			$[a], [b] \mapsto [a+b]$	$[a] \mapsto [-a]$	$[a], w \mapsto [a \cdot w]$	Key switching	Shrinking
[29]	$\mathbb{Z}_{2^k}$	2 GSW ciphertexts	✓	*			
Ours	$\mathbb{Z}_p$, prime p	1 GSW ciphertext	✓	✓	✓	✓	✓

Double-CRT version of GSW FHE schemes implementing single-message bootstrapping, such as [14,12,8], 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 between 2^{32} and 2^{64}. Since all homomorphic operations are defined modulo Q, these schemes can be implemented using native integer 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, e.g., with 32 bits, then uses the Chinese Remainder Theorem (CRT) to express operations modulo Q as independent operations modulo each q_i, allowing the use of native integer types again. As all the polynomials composing ciphertexts are stored in the FFT domain and the FFT can be seen as a type of CRT, this representation is often called Double-CRT [21] or, alternatively, RNS representation [20].

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 formally describe a double-CRT version of the accumulator, i.e. the GSW scheme [4], including all common operations already existing for GSW, such as homomorphic multiplication and external product, and 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, i.e., approximate deomposition in the TFHE scheme [12]. However, in the double-CRT representation, as there is no notion of least significant bits, 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. Reduc-

[4] A double-CRT version of GSW is implemented in the Lattigo library, but there is no formal description and analysis of the scheme. Moreover, it only includes external products.

ing simultaneously d and Q enables a cubic performance improvement in all core homomorphic operations.

We notice that any protocol or scheme that uses GSW can benefit from our new homomorphic operations, thus, this contribution is of independent interest.

Thanks to this low-level description of the GSW scheme, we estimate the cost of our amortized bootstrapping concretely 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.

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 and with polynomial approximation factors with amortized sublinearly many homomorphic operations. We show that our construction is practical, being up to 3.4 times faster than the non-packed approach we tested.

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 [29] is very high level and also that any implementation of their bootstrapping must be far from practical, even if our double-CRT GSW scheme is used, due to all the hidden constants in the asymptotic costs. Thus, one could reasonably wonder if the amortized bootstrapping would ever be practically feasible, and our algorithms together with our implementation provide a positive answer.

2 Preliminaries

For $a_1, ..., a_k, m_1, ..., m_k \in \mathbb{Z}$, with m_i's being pairwise coprime, let $M = \prod_{i=1}^{k} m_i$ and define $\mathsf{CRT}_{m_1,...,m_k}(a_1, ..., a_k)$ 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 $\mathsf{CRT}^{-1}_{m_1,...,m_k}(a) = (a \bmod m_1, ..., a \bmod m_k)$. For an element $a(X)$ of any polynomial ring of the form $\mathbb{Z}[X]/\langle f(X)\rangle$, we extend CRT and 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 $a = \sum_{i=0}^{d} a_i \cdot X^i$, we define the norm of a as the norm of the coefficient vector $(a_0, ..., a_d)$. 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'.

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.

Plain, ring and circulant LWE In the well-known learning with errors problem (LWE) [33] with parameters n, q, and σ, 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 σ.

The ring version of LWE, known as RLWE [27], 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 $b_i := a_i \cdot s + e_i \bmod q$, for some small noise term e_i, 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 [7] 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 $L_Q : \mathcal{R}_Q \to \tilde{\mathcal{R}}_Q$ 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 the message space is restricted to powers of X, that is, if one just encrypts $X^k \in \tilde{\mathcal{R}}$ for $k \in \mathbb{Z}$ [7].

In Section 3.5, we extend the results [7] so that we can also encrypt non-powers of X (under some conditions), as this is needed in our bootstrapping algorithm, especially, to use Galois automorphisms on GSW ciphertexts.

Subgaussian distributions and independence heuristic A random variable X is subgaussian with parameter $\sigma > 0$, in short σ-subgaussian, if for all $t \in \mathbb{R}$ it holds that $\mathbb{E}[\exp(2\pi t X)] \leq \exp(\pi\sigma^2 t^2)$. If X is σ-subgaussian, then $\forall t \in \mathbb{R},\ \Pr[|X| \geq t] \leq 2\exp(-\pi t^2/\sigma^2)$. This allows one to bound the absolute value of X with overwhelming probability. Namely, by setting $t = \sigma\sqrt{\lambda/\pi}$, we see that $\Pr[|X| \geq \sigma\sqrt{\lambda/\pi}] \leq 2\exp(-\pi(s\sqrt{\lambda/\pi})^2/s^2) = 2\exp(-\lambda) < 2^{-\lambda}$. Linear combinations of independently distributed subgaussians are again subgaussians, i.e., given independent σ_i-subgaussian distributions X_i's, then for any $\mathbf{c} = (c_1, ..., c_n) \in \mathbb{R}^n$, it holds that $Y := \sum_{i=1}^n c_i X_i$ is $\left(\sqrt{\sum_{i=1}^n c_i^2 \sigma_i^2}\right)$-subgaussian. We say that a polynomial a is σ-subgaussian if its coefficients are independent σ_i-subgaussian, with $\sigma_i \leq \sigma$. For any $n \in \mathbb{N}^*$, given f equal to $X^n \pm 1$ and $a, b \in \mathbb{Z}[X]/\langle f\rangle$ following subgaussians with parameters σ_a and σ_b, we assume the independence heuristic, i.e. the coefficients of the noise terms of the LWE, RLWE, and CLWE samples appearing in the linear combinations we consider are independent and concentrated, to say that $a \cdot b$ is $(\sqrt{n} \cdot \sigma_a \cdot \sigma_b)$-subgaussian.

Double-CRT (RNS) representation for polynomial arithmetic Homomorphic operations of commonly used FHE schemes are composed of 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 $f(X) = X^N - 1$. 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 ℓ independent operations over $\mathcal{R}_{q_i}$. Moreover, since efficiently multiplying polynomials modulo $f(X)$ requires fast Fourier transforms or number-theoretic transforms (NTT), one goes one step forward and represents elements of $\mathcal{R}_{q_i}$ in the "NTT form", i.e., given $a(X) \in \mathcal{R}_Q$, one stores the matrix $\texttt{Mat}(a) \in \mathbb{Z}^{\ell\times N}$ defined as $\texttt{row}_i(\texttt{Mat}(a)) := \mathsf{NTT}_{q_i}(a(X))$. Notice that we need to choose q_i such that a suitable primitive root of unity $\omega_i \in \mathbb{Z}_{q_i}$ exists.

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

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

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

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

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

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.

Base extension In some situations, we may want to operate on polynomials defined modulo different values Q and D. To do this we need to represent both operands on a common modulus 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 $|u_i| \le 1/2$ 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 $\|u(X)\|_\infty \le 1/2$ [23].

For completeness, we show this operation in detail in Appendix C. It costs $v + w$ NTTs and $O(v \cdot w \cdot N)$ modular multiplications, and it is defined as follows:

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

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, $\mathbf{I}_2$ be the 2×2 identity matrix, and $\otimes$ denote the Kronecker tensor product. 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 $G^{-1} : \mathbb{Z}_Q^2 \to \mathbb{Z}^{2 \times 2d}$ such that, if $\mathbf{A} = G^{-1}(a, b)$, then $\|\mathbf{A}\|_\infty \leq B$ and $\mathbf{A} \cdot \mathbf{G} = (a, b) \mod Q$. 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 $G^{-1}(\mathbf{C}) \in \mathcal{R}^{2d \times 2d}$.

The main example of a gadget matrix is the one defined by some base $B \in \mathbb{Z}$, which corresponds to $d = \lceil \log_B(Q) \rceil$, $\mathbf{g} = (B^0, B^1, ..., B^{d-1})^T$, and quality B. For instance, if $B = 2$, then $G^{-1}(a, b) = (a_0, ..., a_{d-1}, b_0, ..., b_{d-1})$, where a_i's and b_i's are the bits of a and b, respectively.

The CRT-gadget decomposition is of central importance in our double-CRT GSW scheme, presented in Section 3, and it is defined as follows. Let $q_1, ..., q_\ell$ be prime numbers and $Q := \prod_{i=1}^{\ell} q_i$. Let $d \in \mathbb{N}$ be the "number of digits". For simplicity, assume that $d|\ell$ and define $k := \ell/d \in \mathbb{Z}$. Then, for $1 \leq i \leq d$, define the i-th "CRT digit" as $D_i := \prod_{j=(i-1)\cdot k+1}^{i \cdot k} q_i$, that is, a product of k consecutive primes. Finally, define $Q_i := Q/D_i$ and $\hat{Q}_i := (Q/D_i)^{-1} \bmod D_i$. Then the gadget matrix is $\mathbf{G} = \mathbf{I}_2 \otimes \mathbf{g} \in \mathbb{Z}^{2d \times 2}$ where $\mathbf{g} := (Q_1 \cdot \hat{Q}_1, ..., Q_d \cdot \hat{Q}_d)$. More explicitly,

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

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

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 α 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 α modulo Q before decomposing it.

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

$$\mathsf{LWE}^Q_{\mathbf{s}}(\Delta \cdot m, E) := \{(\mathbf{a}, b) \in \mathbb{Z}_Q^{n+1} : b = [\mathbf{a} \cdot \mathbf{s} + e + \Delta \cdot m]_Q \text{ where } e \text{ is } E\text{-subgaussian } \}.$$

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

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

Basically the same definition applies to CLWE ciphertexts:

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

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 produces $e' = e + \Delta \cdot m \bmod Q$, then we output $\lfloor e'/\Delta \rceil \bmod t$. If $\|e' - \Delta \cdot m\| < Q/(2t)$, then the decryption correctly outputs $m \bmod t$.

Common homomorphic operations Generally, FHE schemes allow us to add and multiply ciphertexts homomorphically. In this section we briefly show a list of other common homomorphic operations that apply to RLWE and that will be used here on both RLWE and CLWE ciphertexts. Readers not familiar with them can read Appendix D for a detailed description.

- *Modulus switching*: takes a ciphertext $\mathbf{c} = (a, b) \in \mathsf{RLWE}_{s,Q}(m, E)$, where $Q = \prod_{i=0}^{\ell-1} q_i$, and prime q diving Q, and outputs $\mathbf{c}' \in \mathsf{RLWE}_{s,Q'}(m, E')$, where $Q' = Q/q$ and $E' \leq \sqrt{(E/q)^2 + \|s\|_2^2/2}$. Moreover, it costs 2ℓ NTTs and $O(k\ell N)$ multiplications on $\mathbb{Z}_{q_i}$.
- *Key switching*: takes as input a ciphertext $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta \cdot m, E)$ and a key-switching key $\mathbf{K} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(z,\, E_k)$, both in double-CRT form. It outputs $\mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m, E')$, where $E' \leq O(\sqrt{E^2 + dp \cdot D^2 \cdot E_k^2})$, with $D = \max(D_1, ..., D_d)$. Moreover, it costs $d \cdot \ell$ NTTs and $O(\ell^2 \cdot p)$ products on $\mathbb{Z}_{q_i}$.
- *Automorphism*: it takes $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m, E)$, $u \in \mathbb{Z}_p$, and $\mathbf{K} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s(X^u),\, E_k)$ as input. It outputs $\mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m(X^u), E')$. The noise growth and the cost are the same as the ones of the key switching.

Ring Packing In [29], 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 of $n \cdot L$ RLWE ciphertexts as

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

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

3 Double-CRT GSW encryption scheme

In this section, we formalize a double-CRT version of the GSW scheme supporting all the standard operations, like the 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-LWE problem. 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. In more detail, the set of GSW encryptions m with a scaling factor α is denoted by $\tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m)$. 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. We can write $\tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m, E)$ to specify that $\mathbf{e}$ is E-subgaussian.

Saying that $\mathbf{C}$ is in double-CRT form means that each entry $c_{i,j} \in \tilde{\mathcal{R}}_Q$ is stored as $\mathtt{Mat}(c_{i,j})$, as described in Section 2.

- $\mathsf{GSW.ParamGen}(1^\lambda)$: Choose a prime number p, standard deviations $\sigma_{\mathsf{err}}, \sigma_{\mathsf{sk}} \in \mathbb{R}$, and an integer $Q := \prod_{i=1}^{\ell} q_i$, where $q_1, ..., q_\ell$ are small primes (say, with 32 bits), such that the $(p, Q, \sigma_{\mathsf{err}}, \sigma_{\mathsf{sk}})$-RLWE problem offers us λ 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 $d|\ell$ and let $u := \ell/d \in \mathbb{Z}$. Then, for $1 \le i \le d$, define each "CRT digit" as $D_i := \prod_{j=(i-1)\cdot u+1}^{i\cdot u} q_i$, that is, a product of u consecutive primes. Output $\mathsf{params} = (p, Q, \sigma_{\mathsf{err}}, \sigma_{\mathsf{sk}}, d, \{q_i\}_{i=1}^{\ell}, \{D_i\}_{i=1}^{d})$.
- $\mathsf{GSW.KeyGen}(\mathsf{params})$: Sample $\bar{s}_0, ..., \bar{s}_{p-1}$ following a discrete Gaussian over $\mathbb{Z}$ with parameter σ_{sk}. Let $\bar{s} = \sum_{i=0}^{p-1} \bar{s}_i \cdot X^i$ then project $\bar{s}$ as $s := L((1 - X)\bar{s}) \in \tilde{\mathcal{R}}$. Output $\mathsf{sk} := (s, \bar{s})$.
- $\mathsf{GSW.Enc}(\mu, \mathsf{sk})$: 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 σ_{err}. Output $\mathbf{V} + X^\mu \cdot \mathbf{G} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot X^\mu)$.

Lemma 1 (Security of GSW). *If the decisional $(p, Q, \sigma_{\mathsf{err}}, \sigma_{\mathsf{sk}})$-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 [7]. □

We now present the homomorphic operations that can be performed with GSW.

3.1 Shrinking gadget matrices

We begin by defining a new operation called *shrinking*, whose main purpose is to reduce the size of ciphertexts. Let $Q, Q_i, \hat{Q}_i$, and D be as in Section 2. 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, $D_1, ..., D_k$, remove the two $2k$ rows corresponding to them, and obtain a new gadget matrix with respect to the digits $D_{k+1}, ..., D_d$. However, by doing so, we obtain a scaled gadget matrix $\mathbf{G}_\beta$ with respect to a new scaling factor β.

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

Algorithm 1: Shrink matrix

Input: $\mathbf{C} \in \mathcal{R}_Q^{2d\times 2}$, CRT digits $D_1, ..., D_d$, a scaling factor α, 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. $D^{(k)} := D_1 \cdot ... \cdot D_k$
2. $Q' := Q/D^{(k)}$
3. $\bar{\mathbf{C}} := \pi_k(\mathbf{C})$
4. $\mathbf{C}' := \bar{\mathbf{C}}/D^{(k)} \bmod Q'$
5. $\alpha' := \alpha \cdot \mathsf{CRT}_{D_{k+1},...,D_d}(D^{(k)})^{-1} \bmod Q'$.
6. **return** $\mathbf{C}'$, α'

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 $Q' := Q/(D_1 \cdot ... \cdot D_k)$. Then, given a scaled gadget matrix $\mathbf{G}_\alpha$ with respect to the CRT basis $D_1, ..., D_d$, Algorithm 1 outputs $\mathbf{G}' \in \mathbb{Z}^{2(d-k)\times 2}$ and $\alpha' \in \mathbb{Z}$ such that $\mathsf{CRT}^{-1}(a, b) \cdot \mathbf{G}' = \alpha' \cdot (a, b) \bmod Q'$, for any $(a, b) \in \mathcal{R}_{Q'}^2$, where CRT^{-1} is the decomposition with respect to $D_{k+1}, ..., D_d$.*

Proof. Presented in Appendix F.1. □

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 α. We define the operation $\mathsf{GSW.Shrink}(\mathbf{C}, k)$ essentially by applying Algorithm 1 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 2 and prove its correctness in Lemma 3.

Lemma 3 (Correctness and cost of ciphertext shrinking). *Let $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m, E)$, $k \in \mathbb{N}^*$ such that $k < d$, and $\mathbf{C}', \alpha'$ be the output of Algorithm 2. Assume that s is S-subgaussian for some S. Then, $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha' \cdot m, E')$ with $E' \leq O(E/D^{(k)} + \sqrt{p} \cdot S)$.*

Moreover, assuming that each CRT digit D_i is a product of ℓ/d primes, the cost of Algorithm 2 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, $\mathsf{ModSwt}_{Q\to Q'}(\mathtt{row}_i(\bar{\mathbf{C}}))$

Algorithm 2: Shrink ciphertext

Input: $\mathbf{C} \in \tilde{\mathcal{R}}_Q^{2d \times 2}$ in double-CRT form, scaling factor α, CRT digits $D_1, ..., D_d$, 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: $E \mapsto O(E/D^{(k)} + \sqrt{p} \cdot S)$

$D^{(k)} := D_1 \cdot ... \cdot D_k$

$Q' := Q/D^{(k)}$

$\bar{\mathbf{C}} := \pi_k(\mathbf{C}) \in \tilde{\mathcal{R}}_{Q'}^{2(d-k) \times 2}$

for $1 \leq i \leq 2 \cdot (d-k)$ **do**

  $\mathbf{c}_i := \mathsf{ModSwt}_{Q \to Q'}(\mathtt{row}_i(\bar{\mathbf{C}}))$

Define $\mathbf{C}'$ such that $\mathtt{row}_i(\mathbf{C}') = \mathbf{c}_i$.

$\beta := \mathsf{CRT}_{D_{k+1},...,D_d}(D^{(k)}, ..., D^{(k)})^{-1} \bmod Q'$.

$\alpha' := \alpha \cdot \beta \bmod Q'$

return $\mathbf{C}'$, α'

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 π_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 ℓ costs 2ℓ 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 $\mathbb{Z}_{q_i}$. □

3.3 External product in RNS representation

The external product is a homomorphic multiplication between a CLWE ciphertext and a GSW ciphertext. In our case, there is a little difference because we have to take care of the scaling factor α. Namely, firstly we multiply the CLWE ciphertext by $\alpha^{-1} \bmod Q$ before decomposing it with respect to the scaled gadget matrix $\mathbf{G}_\alpha$, so that the result CLWE ciphertext does not depend on α. We now present this operation in detail in Algorithm 3 and prove its correctness in Lemma 4.

Lemma 4 (Correctness and cost of external product). *On input* $\mathbf{c} \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m_0, E_0)$ *and* $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m_1, E_1)$*, Algorithm 3 outputs* $\mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m_0 \cdot m_1, E')$ *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* $\mathbb{Z}_{q_i}$.

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 E-subgaussian $\mathbf{e}$. Let $\mathbf{u} = (u_1, ..., u_{2d})$ 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 $\mathbf{u} \cdot \mathbf{G}_\alpha = \mathbf{c} \mod Q$.

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 $D := \max(D_1, ..., D_d)$, we have $\mathbf{v} = \mathbf{u} + \mathbf{w}$ where $\|\mathbf{w}\| \leq D/2$.

Firstly, we claim that $\mathbf{w} \cdot \mathbf{G}_\alpha = (0, 0) \mod Q$. Indeed, remember that $Q_i := Q/D_i$, $\hat{Q}_i := Q_i^{-1} \bmod D_i$, $\mathbf{g} = (Q_1 \cdot \hat{Q}_1 \cdot [\alpha]_{D_1}, ..., Q_d \cdot \hat{Q}_d \cdot [\alpha]_{D_d})$ and $\mathbf{G}_\alpha = (\ (\mathbf{g}\ \mathbf{0})^T \ \ (\mathbf{0}\ \mathbf{g})^T\ ) \in \mathbb{Z}^{2d \times 2}$, thus,

Algorithm 3: RNS-friendly External product

Input: $\mathbf{c} \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m_0, E_0)$ and $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m_1, E_1)$ both in double-CRT form.
Output: $\mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m_0 \cdot m_1, E')$
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

```
Denote 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 α^{-1}
for 1 ≤ i ≤ d do
    Let u_i := α^{-1} · a mod D_i
    Let u_{i+d} := α^{-1} · b mod D_i
for 1 ≤ i ≤ d do
    Let v_i := FastBaseExtension(u_i, D_i, Q/D_i) ∈ R̃_Q
    Let v_{i+d} := FastBaseExtension(u_{i+d}, D_i, Q/D_i) ∈ R̃_Q
Let v := (v_1, ..., v_{2d}) ∈ R̃_Q^{2d} ▷ it is already in double-CRT format
c' = v · C ∈ R̃_Q^2
return c'
```

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

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

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 2ℓ modular multiplications. Since each CRT digit D_i has ℓ/d prime factors, each base extension costs ℓ 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[5] takes GSW encryptions of two messages and outputs 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 4. The only caveat is the scaling factor of the output. Both input ciphertexts have scaling factors, say α_0 and α_1, so one could expect that the output would be scaled by $\alpha_0 \cdot \alpha_1$. However, since external products output CLWE encryptions that do not depend on the scaling factor of the GSW ciphertext, the output of the GSW multiplication is scaled only by, say, α_0.

Algorithm 4: RNS-friendly GSW homomorphic multiplication

Input: $\mathbf{C}_0 \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha_0 \cdot m_0, E_0)$ and $\mathbf{C}_1 \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha_1 \cdot m_1, E_1)$ both in double-CRT form.

Output: $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s((\alpha_0) \cdot m_0 \cdot m_1, E)$

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

▷ `Consider` $\Delta_i := Q_i \cdot \hat{Q}_i \cdot \alpha_0 \bmod D_i$

for $1 \leq i \leq d$ **do**
- Let $\mathbf{c}_i := \texttt{row}_i(\mathbf{C}_0) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_i \cdot m_0, E_0)$
- $\mathbf{c}'_i = \mathbf{c}_i \boxdot \mathbf{C}_1 \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_i \cdot m_0 \cdot m_1, E)$ ▷ `External product`

for $d+1 \leq i \leq 2 \cdot d$ **do**
- Let $\mathbf{c}_i := \texttt{row}_i(\mathbf{C}_0) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(-\Delta_i \cdot m_0 \cdot s, E_0)$
- $\mathbf{c}'_i = \mathbf{c}_i \boxdot \mathbf{C}_1 \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(-\Delta_i \cdot m_0 \cdot m_1 \cdot s, E)$ ▷ `External product`

Let $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$ such that $\texttt{row}_i(\mathbf{C}') = \mathbf{c}'_i$

return $\mathbf{C}'$

3.5 Key switching for GSW

If one instantiates the double-CRT GSW scheme over the usual power-of-two cyclotomic ring, i.e. modulo $X^N + 1$, then one can freely choose the keys that will be switched with no issue regarding security. However, we are using the circular ring, $X^p - 1$, and so have to be more careful.

In [7], it is shown that GSW over circulant rings is secure if one just encrypts powers of X. The main problem was that in a circular ring an attacker can interpret an element $a \in \mathcal{R}$ as a polynomial $a' \in \mathbb{Z}[X]$. In this case, it holds that $a' = a + u \cdot (X^p - 1)$ for some $u \in \mathbb{Z}[X]$, and so $a'(1) = a(1) \in \mathbb{Z}$. Thus, a ciphertext evaluated at one would produce the pair $(a(1), b(1) = a(1) \cdot s(1) + \Delta m + e(1)) \in \mathbb{Z}^2$, which could leak information about s or the message.

To solve this Bonnoron, Ducas, and Fillinger [7] apply a function to the RLWE samples that fixes the values of the polynomials when they are evaluated at one, such that $(a(1), b(1)) = (0, \Delta)$, thus are independent of the secret key and leak no information on the message.

However, 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 [7] to show that it is also safe to

[5] In [13], a homomorphic multiplication between two GSW ciphertexts is called *internal product*.

Algorithm 5: GSW key switching

Input: $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_z^d(\alpha \cdot m, E)$, $\mathbf{K} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(z, E_k)$, $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot (-s), E_s)$, all in double-CRT form.
Output: $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m, E')$,
Complexity: $3 \cdot d^2 \cdot \ell$ NTTs and $O(d \cdot \ell^2 \cdot p)$ products on $\mathbb{Z}_{q_i}$.
Noise growth: $E' \in O(\sqrt{dp} \cdot D \cdot E_s + \sqrt{p} \cdot \|s\| \cdot E + p \cdot \sqrt{d} \cdot D \cdot E_k \cdot \|s\|)$

for $1 \leq i \leq d$ **do**
  $\mathbf{c}_i = \mathsf{KeySwt}(\mathtt{row}_i(\mathbf{C}), \mathbf{K}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_i \cdot m)$
  $\mathbf{c}_{d+i} = \mathbf{c}_i \boxdot \mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(-\Delta_i \cdot m \cdot s)$
Let $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$ such that $\mathtt{row}_i(\mathbf{C}') = \mathbf{c}_i$
return $\mathbf{C}'$

encrypt z. The crucial property here is: In the case of key-switching keys, we are only encrypting the secret polynomials of the CLWE problem, which are always of the form $s = L((1 - X) \cdot s')$, and so always result in zero when they are evaluated at one. Therefore, as $m(1) = 0$, it is secure to encrypt them under the CLWE problem. We prove this formally for any constant c such that $m(1) = c$ in Appendix B.

GSW key switching via two-layer reconstruction Given $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_z^d(\alpha \cdot m)$, remember that for $1 \leq i \leq d$, $\mathtt{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta_i \cdot m)$, where $\Delta_i := Q_i \cdot \hat{Q}_i \cdot \alpha_i$. Also, for $d + 1 \leq i \leq 2 \cdot d$, $\mathtt{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_z(-\Delta_i \cdot m \cdot z)$. Thus, to switch $\mathbf{C}$ to a key s, we just need to use the first d rows. Namely, we use the RLWE/CLWE key switching to obtain CLWE 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 5.

From the costs of the external product and of the key switching, 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 $D = \max(D_1, ..., D_d)$. 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 CLWE samples with a 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.

The key switching that we present in this section avoids that "double accumulation" by producing independent CLWE 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. In 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 6, but defer the proof of its correctness to Appendix F.3.

3.6 GSW automorphism

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

Algorithm 6: NoiseReducedGSWKeySwt

Input: $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_z^d(\alpha \cdot m, E)$, $\mathbf{K}_z \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(z,\, E_z)$, $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s,\, E_s)$, and $\mathbf{K}_{sz} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s \cdot z,\, E_{sz})$, all in double-CRT form.
Output: $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m, E')$,
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`
for $1 \le i \le d$ **do**
  $\mathbf{c}_i = \mathsf{KeySwt}(\mathtt{row}_i(\mathbf{C}), \mathbf{K}_z) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_i \cdot m)$

▷ `Now construct the last` d `rows`
for $1 \le i \le d$ **do**
  Let $(a, b) = \mathtt{row}_i(\mathbf{C}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta_i \cdot m)$
  for $1 \le i \le d$ **do**
    ▷ `Each base extension costs` ℓ `NTTs and` $O(p \cdot \ell^2/d)$ `multiplications on` $\mathbb{Z}_{q_i}$
    $h_i = \mathtt{FastBaseExtension}(a \bmod D_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$
    $y_i = \mathtt{FastBaseExtension}(b \bmod D_i, D_i, Q/D_i) \in \tilde{\mathcal{R}}_Q$
  $\mathbf{h} := (h_1, ..., h_d)$
  $\mathbf{y} := (y_1, ..., y_d)$
  $a' := \mathbf{h} \cdot \mathtt{col}_1(\mathbf{K}_{sz}) - \mathbf{y} \cdot \mathtt{col}_1(\mathbf{K}_s)$
  $b' := \mathbf{h} \cdot \mathtt{col}_2(\mathbf{K}_{sz}) - \mathbf{y} \cdot \mathtt{col}_2(\mathbf{K}_s)$
  $\mathbf{c}_{d+i} := (a', b') \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(-\Delta_i \cdot m \cdot s)$
Let $\mathbf{C}' \in \tilde{\mathcal{R}}_Q^{2d \times 2}$ such that $\mathtt{row}_i(\mathbf{C}') = \mathbf{c}_i$
return $\mathbf{C}'$

Notice that the first d rows of $\mathbf{C}$ are regular CLWE encryptions, thus, automorphisms work as usual producing new CLWE encryptions under the key $\eta(s)$. The other d rows can be seen as CLWE encryptions of the $-\Delta_i \cdot m \cdot s$, therefore, the automorphism generates CLWE 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, η does not change the distribution of the noise. In summary, $\eta(\mathbf{C}) \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_{\eta(s)}(\alpha \cdot \eta(m), E)$.

Then, applying GSW key switching on $\eta(\mathbf{C})$ gives us $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot \eta(m), E')$. Hence, the noise growth and the cost are the 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 $X^{\mathbf{u} \cdot \mathbf{v} \bmod p}$.

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 $2 \cdot d \cdot k$ CLWE key switchings, we want to use automorphisms on the CLWE samples so that the cost of k automorphisms also drops to $k \cdot d$, that is, halving the cost compared to GSW automorphisms.

Algorithm 7: $\mathsf{EvalScalarProd}$: Evaluate scalar product in the exponent of X

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

Output: $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot X^y, E')$ 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: $E \mapsto O\left(\sum_{i=1}^k \left(\sqrt{d}Dp \cdot \|s\| \cdot E_i\right) + \sqrt{dp} \cdot D \cdot E_s + \bar{E} + \hat{E}\right)$, where $\bar{E} = \sqrt{p} \cdot \|s\| \cdot E_1$ and $\hat{E} = \sqrt{p} \cdot \|s\| \cdot \sqrt{k+1} \cdot E_{KS}$ with $E_{KS} \in O(\sqrt{dp} \cdot D \cdot E_k)$.

```
▷ Consider that Δ_i := Q_i · Q̂_i · α_i, where α_i = α mod D_i
Define u_{k+1} = 1
for 1 ≤ i ≤ d do
    Let c_i := row_i(C_1) ∈ R̃_Q LWE_s(Δ_i · X^{m_1}, E_1)
    c_i = Auth(c_i, u_1 · u_2^{-1} mod p) ∈ R̃_Q LWE_s(Δ_i · X^{m_1·u_1·u_2^{-1}}, E_1 + E_KS)
    ▷ Let S^(j) := u_{j+1}^{-1} · Σ_{i=1}^{j} m_i · u_i mod p
    for 2 ≤ j ≤ k do
        c_i = c_i ⊡ C_j ∈ R̃_Q LWE_s(Δ_i · X^{S^(j-1) + m_j}, Σ_{i=2}^{j} (√(dp) · D · E_i) + (E_1 + E_KS))
        v = u_j · u_{j+1}^{-1} mod p
        c_i = Auth(c_i, v, K_v) ∈ R̃_Q LWE_s(Δ_i · X^{S^(j)}, Σ_{i=1}^{j} (√(dp) · D · E_i) + E_1 + √(j+1) · E_KS)
▷ Now, construct the other d rows of the GSW sample
for 1 ≤ i ≤ d do
    Let c_{d+i} := c_i ⊡ K_s ∈ R̃_Q LWE_s(−Δ_i · X^{S^(k)} · s, E)
Define C' ∈ R̃_Q GSW_s^d(α · X^{S^(k)}, E) such that row_i(C') = c_i
return C'
```

Hence, given $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot X^{v_i})$, we define $\mathbf{C}'_1 = \mathbf{C}_1$, and for $2 \leq i \leq k$, we want to compute $\mathbf{C}'_i = \mathsf{Auth}(\mathbf{C}_i, u_i) \cdot \mathbf{C}'_{i-1}$.

The main idea is to extract the d rows of $\mathbf{C}'_i$ that correspond to the CLWE samples encrypting X^m for some message m and ignore the other d rows. So, let $\mathbf{c}_j \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_j \cdot X^m)$ 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 [7] 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. $\mathbf{c}'_j = Auth(\mathbf{c}_i, u_i^{-1})$ (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. $\mathbf{c}'''_j = Auth(\mathbf{c}''_i, u_i)$ (encrypts $X^{m + u_i \cdot v_i}$)

Repeating this k times, at the end, we have $\mathbf{c}'''_j \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_j \cdot X^{\sum u_i v_i})$, 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 5.

Since the whole algorithm uses $k \cdot d$ external products and $k \cdot d$ CLWE 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, therefore we are gaining a factor of around 2.33. We show this procedure in detail in Algorithm 7.

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 we can finally replace the homomorphic Nussbaumer transform and the SlowMult algorithm used in [29] by the Number Theoretic Transform (NTT) and a simpler point-wise multiplication.

It is well known that over "cyclic polynomial rings" of the form $R_p := \mathbb{Z}_p[X]/\langle X^N - 1\rangle$, 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 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 $(a(\omega^0), a(\omega^1), ..., a(\omega^{N-1})) \in \mathbb{Z}_p^N$. Now let $\odot : \mathbb{Z}^N \to \mathbb{Z}^N$ be the entrywise multiplication. Then it holds that

$$\mathsf{NTT}^{-1}(\mathsf{NTT}(a) \odot \mathsf{NTT}(z)) \equiv N \cdot a \cdot z \mod \langle X^N - 1, p\rangle.$$

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 ψ^{-1}. In more detail, let $\boldsymbol{\psi} := (\psi^0, \psi, ..., \psi^{N-1})$ and $\boldsymbol{\psi}^{-1} := (\psi^0, \psi^{-1}, ..., \psi^{-(N-1)})$, where ψ is a $2N$'th root of unity as defined above, then

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

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 $2N$. Also, given ψ, 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 ρ 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 the complexity $O(N \cdot \log N)$. However, because the noise overhead of the homomorphic NTT is proportional to N^ρ, we restrict ourselves to instantiating the algorithm with small values of ρ only.

We show the radix-m inverse NTT in detail in Algorithm 8, where the multiplication by $\boldsymbol{\psi}^{-1}$ 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 a k-th root of unity in $\mathbb{Z}_p$ by w_k. At the beginning of the algorithm, we start with w_N, then all the others roots of unity that appear 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 $T(N) =$

Algorithm 8: NTT_m^{-1} - Inverse NTT in time $O\left(\rho \cdot N^{1+\frac{1}{\rho}}\right)$

Input: $(f_0, \ldots, f_{N-1}) \in \mathbb{Z}_p^N, \rho \in \mathbb{Z}^+$, $\tilde{\rho} \in \mathbb{Z}^+$, where $\tilde{\rho}$ starts at 1, $m \in \mathbb{Z}^+$ s.t. $m \mid N$
Output: $\mathsf{NTT}^{-1}(\mathbf{f})$
1 **if** $\rho = 1$ **then**
2 $\quad \mathbf{u} = \boldsymbol{\psi}^{-1} N^{-1} \bmod p$
3 **else**
4 $\quad \mathbf{u} = (1, 1, ..., 1) \in \mathbb{Z}^N$
5 **if** $\tilde{\rho} = \rho$ **then**
$\quad \triangleright$ `Trivial quadratic algorithm, time` $O\left(N^2\right)$
6 $\quad$ **for** $0 \leq j < N$ **do**
7 $\quad\quad a_j = \sum_{i=0}^{n-1} f_i \cdot u_j \cdot w_N^{-i \cdot j} \bmod p$
8 **else**
$\quad \triangleright$ `General case with recursive calls`
9 $\quad$ **for** $0 \leq i < m$ **do**
10 $\quad\quad \mathbf{g} = \left(f_i,\, f_{m+i},\, ...,\, f_{m(N/m-1)+i}\right)$
11 $\quad\quad \mathbf{h}^{(i)} = \mathsf{NTT}_m^{-1}(\mathbf{g}, \tilde{\rho} + 1)$
12 $\quad$ **for** $0 \leq k_1 < \frac{N}{m}$ **do**
13 $\quad\quad$ **for** $0 \leq k_2 < m$ **do**
14 $\quad\quad\quad j = k_1 + \frac{N}{m} \cdot k_2$
15 $\quad\quad\quad a_j = \sum_{i=0}^{m-1} h_{k_1}^{(i)} \cdot u_j \cdot w_N^{-i \cdot k_1} \cdot w_m^{-i \cdot k_2} \bmod p$
16 **return** $(a_0, ..., a_{N-1})$

$m \cdot T(N/m) + m \cdot N$. Iterating it ρ 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 use the quadratic algorithm, 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:

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

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

Lemma 5 (Time complexity in terms of homomorphic operations). *The homomorphic evaluation of the inverse NTT, Algorithm 8, 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 automorphism 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 8, 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 7, we can have a concrete instead of asymptotic estimation of the number of NTTs.

Lemma 6 (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 8 with recursive level ρ, dimension N, and with lines 7 and 15 implemented with the* EvalDotProduct*, Algorithm 7. If no ciphertext shrinking is used, then the total number of NTTs is*

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

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

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

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

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 7. A detailed proof is provided in Appendix G. □

Error growth of the homomorphic inverse NTT Assuming again that the sums in Algorithm 8 are implemented with the $\mathsf{EvalScalarProd}$ (Algorithm 7), we have the following result.

Lemma 7. *Consider the homomorphic evaluation of Algorithm 8 on input $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot X^{f_i}, E)$, where $0 \leq i < n$. Let $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot (-s), E_s)$ and $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s(X^v), E_k)$ be the keys to compute the sums with Algorithm 7. 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 ρ is the chosen recursive depth.

Proof. We present the proof in Appendix F.4. □

4.2 Partial decryption using NTT_m^{-1}

Remember that to decrypt $(a, b) \in \hat{\mathcal{R}}_p\mathsf{LWE}_z(m, E)$, we have to compute $b^\star := b - a \cdot z \bmod p$, then it holds that $b^\star = e + \Delta \cdot m$, where $\Delta = \lfloor p/t \rceil$. Then we can recover each coefficient m_i by taking the $\log t$ most significant bits. This last step is message extraction and we present it in Section 4.3. In this section, we present an algorithm that uses the homomorphic inverse NTT to compute $b^\star$.

The first part of the algorithm computes the forward NTT of a and of b, to obtain $\bar{\mathbf{a}} := \mathsf{NTT}(\boldsymbol{\psi} \odot a)$, $\bar{\mathbf{b}} := \mathsf{NTT}(\boldsymbol{\psi} \odot b)$, and the bootstrapping key encrypting $\bar{\mathbf{z}} := \mathsf{NTT}(-\boldsymbol{\psi} \odot z)$, we use GSW automorphisms to compute the entrywise product $\bar{\mathbf{a}} \odot \bar{\mathbf{z}} = (\bar{a}_0 \cdot \bar{z}_0, ..., \bar{a}_{N-1} \cdot \bar{z}_{N-1})$. At this point, we also add $\mathsf{NTT}(\boldsymbol{\psi} \odot b)$. Finally, we apply the inverse NTT homomorphically to obtain GSW ciphertexts encrypting $b^\star$.

Algorithm 9: NTTDec, the homomorphic partial decryption

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

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

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

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

Parse $\mathbf{c}$ as $(a, b) \in \hat{\mathcal{R}}_p^2$ where $\hat{\mathcal{R}}_p = \mathbb{Z}_p[X]/\langle X^N + 1\rangle$

$(\bar{a}_0, ..., \bar{a}_{N-1}) \leftarrow \boldsymbol{\psi} \odot \mathsf{NTT}(a)$; ▷ $\mathsf{NTT}(a) \in \mathbb{Z}_p^N$

$(\bar{b}_0, ..., \bar{b}_{N-1}) \leftarrow \boldsymbol{\psi} \odot \mathsf{NTT}(b)$; ▷ $\mathsf{NTT}(b) \in \mathbb{Z}_p^N$

for $i \in \{0, \ldots, N-1\}$ **do**

  $\bar{\mathbf{K}}_i = \eta_{\bar{a}_i}(\mathbf{K}_i)$; ▷ $\bar{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_{\eta_{\bar{a}_i}(s)}^d(1 \cdot X^{-\bar{a}_i \cdot \bar{z}_i}, E)$

  $\mathsf{KS}_{\eta_{\bar{a}_i}(s) \to s}(\bar{\mathbf{K}}_i)$; ▷ $\bar{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot X^{-\bar{a}_i \cdot \bar{z}_i}, E')$

  $\tilde{\mathbf{K}}_i = X^{\bar{b}_i} \cdot \bar{\mathbf{K}}_i$; ▷ $\tilde{\mathbf{K}}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot X^{\bar{b}_i - \bar{a}_i \cdot \bar{z}_i}, E')$

$(\bar{\mathbf{C}}_0, ..., \bar{\mathbf{C}}_{N-1}) = \mathsf{NTT}^{-1}(\tilde{\mathbf{K}}_0, \ldots, \tilde{\mathbf{K}}_{N-1})$; ▷ $\bar{\mathbf{C}}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot X^{e_i + \Delta \cdot m_i}, E'')$

Lemma 8 (Correctness, cost, and noise overhead of NTTDec). *Given a ciphertext $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_p\mathsf{LWE}_z(\Delta \cdot m, E^{(in)})$, the bootstrapping keys $\mathbf{K}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot X^{\bar{z}_i}, E)$, where $(\bar{z}_1, \ldots, \bar{z}_{N-1}) = \mathsf{NTT}_p(\boldsymbol{\psi} \odot -z)$, and the keys used by Algorithm 7, namely $\mathbf{K}_v \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s(X^v), E_k)$ for all $v \in \mathbb{Z}_p^*$ and $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(1 \cdot (-s), E_s)$, where E_s is a constant, then Algorithm 9 outputs GSW encryptions of $X^{e_i + \Delta \cdot m_i}$ with E''-subgaussian noise, where*

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

Moreover, Algorithm 9 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. Let $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_p\mathsf{LWE}_z(m, E^{(in)})$, 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. These operations do not contribute to the complexity, since the homomorphic operations dominate them.

In lines 5 and 6, we compute $\bar{\mathbf{K}}_i$ via GSW automorphism. Since this is done for each i, it 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 5, 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 6, 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 7, 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 9, we obtain ciphertexts of the form $\tilde{\mathbf{C}} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha \cdot X^{\Delta \cdot m + e})$, and we want to extract the message m from the exponent, $X^{\Delta \cdot m+e} \mapsto m$. This message extraction procedure was introduced in [14] and adapted to different settings in subsequent works [12,32,8]. Usually, it is assumed to work with negacyclic rings, however in Algorithm 10 we provide a version of the message extraction algorithm adapted to the cyclic ring. Here we assume messages in $\mathbb{Z}_t$ instead of in $\{0, 1\}$, and apply a programmable bootstrapping, i.e., mapping $X^{\Delta \cdot m+e}$ to $f(m)$ for any given function $f : \mathbb{Z}_t \to \mathbb{Z}_t$. Moreover, our algorithm also takes care of the scaling factor α in the gadget matrix. We prove its correctness and time complexity Appendix F.2.

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 $\mathbf{c}_i \in \mathsf{LWE}^Q_{\mathbf{s}}(\Delta \cdot m_i) \in \mathbb{Z}^{p+1}$, then packs them into one RLWE ciphertext $\mathbf{c} \in \hat{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta \cdot m(X))$, 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 11.

Algorithm 10: $\mathsf{MsgExtract}$

Input: $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha \cdot X^{\Delta \cdot m + e}, E)$, where $\Delta = \lfloor p/t \rceil$ and $|e| < \Delta/2$. A function $f : \mathbb{Z}_t \to \mathbb{Z}_t$

Output: $\bar{\mathbf{c}} \in \mathsf{LWE}^Q_{\mathbf{s}}(\lfloor Q/t \rceil \cdot f(m), E') \in \mathbb{Z}_Q^{p+1}$

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

Noise growth: $E \mapsto O(\sqrt{dp} \cdot D \cdot E)$

Let $t(X) = X^{\Delta/2} \cdot \sum_{i=0}^{t-1} \sum_{j=0}^{\Delta-1} f(i) X^{p - i\Delta - j} \bmod X^p - 1$

Let $\bar{\mathbf{c}} = (0, \lfloor Q/t \rceil \cdot t(X)) \in \tilde{\mathcal{R}}_Q^2$ be a trivial and noiseless encryption of $t(X)$

$\mathbf{c}' := (a', b') = \mathbf{c} \boxdot \mathbf{C}$; $\qquad \triangleright\ \mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\lfloor Q/t \rceil \cdot t(X) \cdot X^{\Delta \cdot m + e}, E')$

Let $\mathbf{A} \in \mathbb{Z}^{p \times p}$ be the circulant matrix of a'

Let $\mathbf{b} \in \mathbb{Z}^p$ be the coefficient vector of b'

Let $\mathbf{u} := (1, 0, ..., 0) \in \{0, 1\}^p$

return $\bar{\mathbf{c}} = [\mathbf{u} \cdot \mathbf{A}, \mathbf{u} \cdot \mathbf{b}]$; $\qquad \triangleright\ \bar{\mathbf{c}} \in \mathsf{LWE}^Q_{\mathbf{s}}(\lfloor Q/t \rceil \cdot f(m), E')$

Algorithm 11: $\mathsf{Bootstrap}$ — for plaintext space $\mathbb{Z}_t$

Input: $\mathbf{c}_i \in \mathsf{LWE}^Q_{\mathbf{s}}(\Delta \cdot m_i, E^{(in)}) \in \mathbb{Z}^{p+1}$ for $0 \le i < N$, where $\Delta = \lfloor Q/t \rceil$. All the bootstrapping and key-switching keys used in Algorithm 9. Arbitrary functions $f_i : \mathbb{Z}_t \to \mathbb{Z}_t$, for $0 \le i < N$.

Output: $\mathbf{c}'_i \in \mathsf{LWE}^Q_{\mathbf{s}}(\Delta \cdot f_i(m_i)) \in \mathbb{Z}^{p+1}$ for $0 \le i < N$.

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

Noise growth: $E^{(in)} \mapsto O\left((p \cdot \sqrt{d} \cdot D)^{\rho+1} \cdot \|s\|^\rho \cdot \sqrt{N} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right)$

$(a^{(1)}, b^{(1)}) = \mathsf{PackLWE}(\mathbf{c}_0, ..., \mathbf{c}_{N-1})$; $\qquad \triangleright\ (a^{(1)}, b^{(1)}) \in \hat{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta \cdot m(X), E^{(1)})$

$(a^{(2)}, b^{(2)}) = \mathsf{ModSwitch}_{Q \to p}(a^{(1)}, b^{(1)})$; $\qquad \triangleright\ (a^{(2)}, b^{(2)}) \in \hat{\mathcal{R}}_p\mathsf{LWE}_z(\Delta' \cdot m(X), E^{(2)})$

$(\mathbf{C}_0, \ldots, \mathbf{C}_{N-1}) = \mathsf{NTTDec}(a^{(2)}, b^{(2)})$; $\qquad \triangleright\ \mathbf{C}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha \cdot X^{e'_i + \Delta' \cdot m_i}, \bar{E})$

for $0 \le i < N$ **do**

$\quad \mathbf{c}'_i = \mathsf{MsgExtract}(\mathbf{C}_i, f_i)$; $\qquad \triangleright\ \mathbf{c}'_i \in \mathsf{LWE}^Q_{\mathbf{s}}(f_i(m_i), E')$

return $\mathbf{c}'_0, ..., \mathbf{c}'_{N-1}$

Lemma 9. *Given at most N LWE ciphertexts and the keys described in Lemma 8, Algorithm 11 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 of Algorithm 11 is asymptotically dominated by the NTTDec, thus, it follows directly from Lemma 8.

Again from Lemma 8, 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 11, the final noise is then $E' = O(\sqrt{dp} \cdot D \cdot \bar{E})$. Hence, it holds that $E' = O\left((p \cdot \sqrt{d} \cdot D)^{\rho+1} \cdot \|s\|^{\rho} \cdot \sqrt{N} \cdot (E \cdot \|s\| + E_k \cdot \sqrt{d} \cdot D)\right)$

□

Corollary 2. *The bootstrapping algorithm presented in Algorithm 11 has noise overhead of $\tilde{O}(\lambda^{1.5+\rho})$, where λ is the security parameter.*

Proof. For a security level of λ 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 λ, let $Q = \tilde{O}(\lambda^{2.5+\rho})$ and consider that the input ciphertexts $\mathbf{c}_i \in \mathsf{LWE}^Q_{\mathbf{s}}(\Delta \cdot m_i, E^{(in)}) \in \mathbb{Z}^{p+1}$ satisfy $E^{(in)} = O(Q/\lambda)$. Then, with probability $1 - 2^{-\lambda}$, the output of Algorithm 11 is correct,* i.e., *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, the parameter $E^{(1)}$ shown in Algorithm 11 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, 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 $\mathsf{MsgExtract}$ is satisfied, i.e., the restriction $\|e\| < \Delta/2$ described in Lemma 11, since there $\Delta = \lfloor p/t \rceil = O(p)$. 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.

□

Remark 1. One can trivially gain a factor $\sqrt{\lambda}$ in the noise presented in Corollary 2 by partially merging $\mathsf{MsgExtract}$ with the homomorphic inverse NTT, Algorithm 8, by starting with encryptions of $\lfloor Q/t \rceil \cdot t(X)$ instead of encryptions of one. This does not change the noise growth of the inverse NTT and we just needs to execute lines 4 to 7 of $\mathsf{MsgExtract}$, which also do not change the noise. With this, the final noise in Corollary 2 is improved to $\tilde{O}(\lambda^{1+\rho})$. This optimization is applied in our implementation.

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 [29] 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. We implemented our scheme mostly from scratch using Intel HEXL Library [6] as the arithmetic backend to provide fast polynomial multiplications. We benchmarked our implementation in a `m6i.metal` instance on AWS (Intel Xeon 8375C at 3.5GHz with 512 GB of RAM at 3200 MHz). We use Ubuntu 22.04 and G++ 11.3.0. Compiling options and further details are available in our repository. All parameter sets presented in this section consider the 128-bit security level estimated using the Lattice Estimator [2] with the default (MATZOV [28]) cost model.

5.1 Parameter selection

In general, bootstrapping capabilities are defined by three parameters: The output dimension p, the input error σ_{in}, and the output error σ_{out} of a bootstrapping. Specifically, a k-bit message is correctly bootstrapped with probability $erf\left(\frac{p/2^{k+1}}{\sigma_{in}\sqrt{2}}\right)$, where erf is the Gauss error function. Let p^* be the modulus of the bootstrapping output, and N be the RLWE input dimension, Equation 4 estimates the value of σ_{in} for the amortized bootstrapping considering a composed circuit (*i.e.*, when we bootstrap the result of a previous bootstrapping). The first term represents noise introduced by the previous bootstrapping and arithmetic operations. The second term estimates the noise introduced by the key switching from dimensions p to N and the ring packing. These first two terms are scaled down by $\frac{p}{p^*}$ by the modulus switching procedure, which, in turn, introduces some rounding noise, represented by the third term in the equation. Compared to a typical bootstrapping, our amortized method has the additional restriction that there must exist a $2N$-th root of unity modulo p. Therefore, we select $p = 12289$ for bootstrapping RLWE samples of dimension $N = 1024$. As the rounding noise is the only one to be introduced directly over modulus p, it is the main limiting factor for our bootstrapping capabilities, and we minimize it by working with sparse ternary keys of Hamming Weight 256.

$$\sigma_{in}^2 \leq \left(\sigma_{out}^2 + (p + N^2)\log_2(p^*)\sigma_{ks}^2\right)\left(\frac{p}{p^*}\right)^2 + \frac{\|s\|_2^2}{12} \tag{4}$$

5.2 Performance of Amortized Bootstrapping

INTT Performance The homomorphic evaluation of the INTT, Algorithm 8, is the core and most expensive procedure in our amortized bootstrapping. Table 3 shows its execution time for $\ell = d = 4$ and $\rho = 2$, with and without shrinking. We choose these parameters only for benchmarking the INTT implementation and shrinking technique. They are not optimized for message bootstrapping. We note that increasing ρ would multiply the output noise by at least $\sqrt{d}pD\,\|s\|$ (Lemma 7), which, in turn, would cubically deteriorate performance on the basic arithmetic. While our estimates and

Table 3: Execution time, in milliseconds, for the INTT for $\ell = 4$ and $\rho = 2$.

p	n	Without shrinking		With shrinking		Shrinking
		Exec. Time	Amortized Time	Exec. Time	Amortized Time	Speedup
12289	512	1,078,033	2,106	695,761	1,359	1.5
12289	1024	3,546,423	3,463	2,132,504	2,083	1.7

Table 4: Execution time, in milliseconds, for the amortized bootstrapping.

p	n	$\ell = d$	Total Time	Amortized Time
12289	1024	3	871,827	851
		4	1,540,075	1,504

experimental data suggest the asymptotic improvements of $\rho = 2$ are greater than the arithmetic overhead, this does not seem to be the case for larger values of ρ.

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 CLWE samples in the message extraction phase. A more efficient way of implementing it is to run the last recombination step (Lines 12 to 15 in Algorithm 8) already using a CLWE as the accumulator for the summations, replacing GSW multiplications with external products. Further, we can also initialize the accumulator with the test vector at the beginning of this last recombination (similarly as introduced in [12]), which simplifies the message extraction and reduces the output error, as explained in Remark 1. This also allows us to use a smaller modulus, with $\ell = 3$. Table 4 shows the results for complete bootstrapping, including key switchings, with $\rho = 2$ and $p^* = 2^{24}$. The failure probabilities for 7 and 8-bit messages are $2^{-62.4}$ and $2^{-17.2}$, respectively. The parameters $\ell = 4$ and $\ell = 3$ are similar in terms of bootstrapping capabilities, but using $\ell = 4$ reduces σ_{out} significantly (in up 2^{49} times for our parameters), enabling variations of our bootstrapping as we exemplify in Section 5.3.

5.3 Comparison with other bootstrapping methods

As the output of the bootstrapping is a set of LWE ciphertexts defined modulo Q, one can classify the bootstrapping algorithms in two categories depending on the size of Q as follows: (1) when Q is small enough to fit into native integer types; (2) when Q is large and double-CRT techniques are needed. The first scenario yields the fastest bootstrapping algorithms known until now, as the ones implemented in TFHE-RS [1], but it is less general, since the output ciphertexts have little noise budget and all the subsequent homomorphic evaluation is done via programmable bootstrapping. The second scenario is more versatile, since we can increase Q to obtain refreshed ciphertexts with more noise budget so that we can compute on them without using programmable bootstrapping. In particular, some constructions require large Q. For example, if one uses scheme switchings, such as Chimera [9], where the slots of a BGV ciphertext are extracted into individual LWE ciphertexts, then functional bootstrapping is applied on each of them, then one needs the output to be defined modulo a large Q so that it is possible to pack the LWE ciphertexts again into a BGV ciphertext. As another example, the high-precision CKKS bootstrapping presented in [22] also requires large Q. Thus, we divide our comparison in those two cases.

Comparison with small-Q bootstrapping algorithms Our amortized bootstrapping produces ciphertexts with large Q, hence this comparison is complicated, since these two types of bootstrapping offer different capabilities, and unfair, since using small Q allows one to implement the bootstrapping using native integer types, which represents a much faster arithmetic back end compared to double-CRT. Nonetheless, this section contains some discussion about this comparison. The state-of-the art implementation of programmable bootstrapping is TFHE-RS [1]. To achieve their impressive performance, they rely on binary secrets, on an FFT library specially tailored for negacyclic rings used in RLWE, and on small (word-sized) modulus. These are all particularities not shared by our scheme. Furthermore, we must note that the performance of such library is the result of several years of extensive research on techniques, parameters, and implementation optimizations [4]. Whereas, the goal of our implementation is just to provide a the first proof of concept of an entirely novel bootstrapping method, and further research on optimal choice of parameters or implementation optimizations are out of our scope.

All that said, we notice that TFHE-RS bootstraps 8-bit messages in 828.1 ms [1], which is only 3% faster than our amortized time in the same machine.

Comparison with large-Q bootstrapping algorithms To have a comparable non-amortized version of our bootstrapping, we implemented, using double-CRT, the algorithm defined in [24], which is the latest bootstrapping method for TFHE-like schemes. We start with the implementation of the accumulator using the double-CRT CLWE variant of the GSW scheme described in Section 3. We estimate noise using Equation 4, but consider $N = 0$ as we do not have the ring packing in this version. We choose parameters that enable similar bootstrapping capabilities for both methods and optimize performance by minimizing the dimension n. The main restriction for this process is the security level of the input. Specifically, we work with sparse ternary keys to minimize the noise, and we need to perform a key switching from dimensions $p = 12289$ to n, which requires an output modulus of at least $p^* \approx 2^{21}$ to accommodate the noise. Considering this, $n = 896$ is the smallest dimension for which the input ring is secure. We could lower n by using Gaussian keys with a larger distribution σ_s, but it is advantageous to avoid the quadratic impact of σ_s compared to the linear impact of n in the square norm of s (third term of Equation 4).

Thus, considering $n = 896$, $\ell = d = 2$, and $p = 12289$, the running time we obtained was around 2.9 seconds, which is 3.4 times slower than our best amortized running times presented in Table 4. The failure probabilities for 7 and 8-bit messages are $2^{-58.5}$ and $2^{-16.3}$, respectively.

5.4 Further Improvements

Since our results are comparable with the state-of-the-art programmable bootstrapping with small ciphertext modulus Q (even though our amortized bootstrapping uses large Q, and thus, needs double-CRT), and faster than non-amortized bootstrapping with large Q, we believe that we achieved the goal of showing that efficient amortized bootstrapping is possible and can be practical. Nonetheless, our implementation is a proof of concept and there are many promising techniques that could be applied to further improve performance. For example, using parameters from Table 4, our implementation requires 76.5 GiB of memory (for $\ell = 3$). However, considering only already existing techniques from the literature (*e.g.*, decomposed automorphisms [24]), we could reduce it to less than 10 GiB of memory. We discuss more about possible further improvements in Appendix H.

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 of Aarhus University. He is supported by the São Paulo Research Foundation under grants 2013/08293-7, 2019/12783-6, and 2021/09849-5.

References

1. zama-ai/tfhe-rs (May 2023), https://github.com/zama-ai/tfhe-rs, original-date: 2022-10-13T09:20:07Z
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046 (2015), https://eprint.iacr.org/2015/046
3. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_17
4. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter optimization & larger precision for (T)FHE. Cryptology ePrint Archive, Report 2022/704 (2022), https://eprint.iacr.org/2022/704
5. Bluestein, L.: A linear filtering approach to the computation of discrete fourier transform. IEEE Transactions on Audio and Electroacoustics **18**(4), 451–455 (1970). https://doi.org/10.1109/TAU.1970.1162132
6. Boemer, F., Kim, S., Seifu, G., de Souza, F.D., Gopal, V.: Intel HEXL: Accelerating homomorphic encryption with intel AVX512-IFMA52. Cryptology ePrint Archive, Report 2021/420 (2021), https://eprint.iacr.org/2021/420
7. Bonnoron, G., Ducas, L., Fillinger, M.: Large FHE gates from tensored homomorphic accumulator. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 18. LNCS, vol. 10831, pp. 217–251. Springer, Heidelberg (May 2018). https://doi.org/10.1007/978-3-319-89339-6_13
8. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: Final: Faster fhe instantiated with ntru and lwe. In: Advances in Cryptology – ASIACRYPT 2022. Springer International Publishing, Cham (2022)
9. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020). https://doi.org/doi:10.1515/jmc-2019-0026, https://doi.org/10.1515/jmc-2019-0026
10. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in LWE-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 1–13. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_1
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (Jan 2020). https://doi.org/10.1007/s00145-019-09319-x
14. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
15. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144

16. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
17. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_28
18. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
19. Guimarães, A., Pereira, H.V.L., Leeuwen, B.v.: Amortized Bootstrapping Revisited: Simpler, Asymptotically-faster, Implemented (2023), https://eprint.iacr.org/2023/014, report Number: 014
20. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 83–105. Springer, Heidelberg (Mar 2019). https://doi.org/10.1007/978-3-030-12612-4_5
21. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_25
22. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for RLWE-based homomorphic encryption. Cryptology ePrint Archive, Report 2021/691 (2021), https://eprint.iacr.org/2021/691
23. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 608–639. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_21
24. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: 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
25. Liu, F.H., Wang, H.: Batch bootstrapping i: A new framework for simd bootstrapping in polynomial modulus. In: Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. p. 321–352. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30620-4_11, https://doi.org/10.1007/978-3-031-30620-4_11
26. Liu, F.H., Wang, H.: Batch bootstrapping ii: Bootstrapping polynomial modulus only requires $\tilde{\mathrm{O}}(1)$ fhe multiplications amortization. In: Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. p. 353–384. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30620-4_12, https://doi.org/10.1007/978-3-031-30620-4_12
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
28. MATZOV: Report on the Security of LWE: Improved Dual Lattice Attack (Apr 2022). https://doi.org/10.5281/zenodo.6412487, https://doi.org/10.5281/zenodo.6412487
29. Micciancio, D., Sorrell, J.: Ring packing and amortized FHEW bootstrapping. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) ICALP 2018. LIPIcs, vol. 107, pp. 100:1–100:14. Schloss Dagstuhl (Jul 2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.100
30. Micheli, G.D., Kim, D., Micciancio, D., Suhl, A.: Faster Amortized FHEW bootstrapping using Ring Automorphisms (2023), https://eprint.iacr.org/2023/112, report Number: 112
31. Nussbaumer, H.: Fast polynomial transform algorithms for digital convolution. IEEE Transactions on Acoustics, Speech, and Signal Processing **28**(2), 205–215 (1980). https://doi.org/10.1109/TASSP.1980.1163372, https://ieeexplore.ieee.org/document/1163372
32. Pereira, H.V.L.: Bootstrapping fully homomorphic encryption over the integers in less than one second. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 331–359. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75245-3_13
33. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603

A Glossary of parameters used in this work

Parameter	Description	Size
λ	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.	$O(\lambda)$
ρ	Recursive depth of the bootstrapping algorithm	–
q_i	Prime factor of ciphertext modulus Q	$< 2^{50}$
Q	Ciphertext modulus defined as $\prod_{1=1}^{\ell} q_i$	$\tilde{O}(\lambda^{1.5+\rho})$
ℓ	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. ℓ/d primes.	$O(q_i \cdot \ell/d)$
D	Upper-bound on the size of the CRT digits	$\max(D_1, ..., D_d)$

B Generalization of circulant-LWE encryption

In [7], 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 10. *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 σ, 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 [7], 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 $S_{Q,p} \times (S_{Q,p} + \Delta m) = S_{Q,p}^2$. □

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 6, we also need to encrypt a product $s \cdot z$, but both keys are also CLWE secrets, thus, we still have $s(1) \cdot z(1) = 0$.

C Algorithm to perform fast base extension

Algorithm 12: 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`

for $1 \leq j \leq w$ **do**
  Let $\mathbf{a}^{(j)} := \mathtt{row}_j(\mathtt{Mat}(a)) \in \mathbb{Z}^N$
  $a^{(j)} := \mathsf{NTT}_{d_j}^{-1}(\mathbf{a}^{(j)}) \in \mathcal{R}_{d_j}$
for $1 \leq i \leq v$ **do**
  $a^{(w+i)} := 0 \in \mathcal{R}_{p_i}$
  for $1 \leq j \leq w$ **do**
    $tmp := a^{(j)} \cdot \hat{D}_j \bmod d_j$
    $tmp := tmp \cdot (D/d_j) \bmod p_i$
    $a^{(w+i)} = (a^{(w+i)} + tmp) \bmod p_i$
for $1 \leq i \leq v$ **do**
  $\mathbf{a}^{(w+i)} := \mathsf{NTT}_{p_i}(a^{(w+i)}) \in \mathbb{Z}_{p_i}^N$
Return $(\mathbf{a}^{(1)}, ..., \mathbf{a}^{(w+v)}) \in \mathbb{Z}^{(w+v)\times N}$.

D Common homomorphic operations

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

$$\mathbf{c}' := \left[\lfloor \mathbf{c} \cdot Q'/Q \rceil\right]_{Q'} = ([\lfloor a/q \rceil]_{Q'}, [\lfloor b/q \rceil]_{Q'}).$$

Generally, $\mathbf{c}'$ is an RLWE encryption of the same message m, but with ciphertext modulus Q' and noise close to e/q. Because rounding is not compatible with double-CRT representation we avoid it by first subtracting $[\mathbf{c}]_q$ 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 $\mathbf{c}' := [\hat{\mathbf{c}}/q]_{Q'} = ((a - \delta_a) \cdot q^{-1}, (b - \delta_b) \cdot q^{-1}) \bmod Q'$.

To subtract δ_a from a we first need to use `FastBaseExtension` from q to Q' to obtain δ_a in base Q. The same is needed for b. Each `FastBaseExtension` costs ℓ 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 $O(N \cdot (\ell - k))$ integer modular multiplications. Thus, the overall number of operations 2ℓ NTTs and $O(k\ell N)$ multiplications on $\mathbb{Z}_{q_i}$.

Algorithm 13: $\mathsf{ModSwitch}$ Algorithm

Input: $\mathbf{c} = (a, b) \in \mathsf{RLWE}_{s,Q}(m)$, Q' such that $Q'|Q$
Output: $\mathbf{c}' \in \mathsf{RLWE}_{s,Q'}(m)$
Complexity: 2ℓ NTT, $O(k\ell N)$ multiplications on $\mathbb{Z}_{q_i}$
Noise growth: $E \to O(E/q + \sqrt{N} \cdot S)$

$\delta_a = \mathsf{FastBaseExtension}_{Q',q}(a \mod q)$
$\delta_b = \mathsf{FastBaseExtension}_{Q',q}(b \mod q)$
$\hat{\mathbf{c}} = (a - \delta_a, b - \delta_b)$
$\mathbf{c}' = [\hat{\mathbf{c}}/q]_{Q'} = \left((a - \delta_a) \cdot q^{-1}, (b - \delta_b) \cdot q^{-1}\right) \mod Q'$

After modulus switching, there is a ϵ 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 δ_a and δ_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. 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 into two steps, where the first step uses the secret key to generate a *public* key-switching key, and the second step consists of using the key-switching key to actually perform the key switching.

We denote by $\tilde{\mathcal{R}}_Q\mathsf{KS}^d_s(z, E)$ the set of key-switching keys from a key z to a key s, having E-subgaussian noise. Given s and z, we generate $\mathbf{K} \in \tilde{\mathcal{R}}_Q\mathsf{KS}^d_s(z, E)$ 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 $\mathbf{g} = (Q_1 \cdot \hat{Q}_1, \ldots, Q_d \cdot \hat{Q}_d)$ is the CRT gadget vector, that is, for $1 \leq i \leq d$, $\mathtt{row}_i(\mathbf{K}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta_i \cdot z, E)$, where $\Delta_i := Q_i \cdot \hat{Q}_i$.

To key switch a ciphertext, $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta \cdot m)$, 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 14.

Algorithm 14: Key switching

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

D.1 Automorphism

Given a ciphertext $\mathbf{c} = (a, b) \in \hat{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m)$ 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 15. 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 $X^{u \cdot v}$.

Algorithm 15: Automorphism

Input: $\mathbf{c} = (a, b) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m, E)$, $u \in \mathbb{Z}_p$, and $\mathbf{K} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s(X^u), E_k)$. Both $\mathbf{c}$ and $\mathbf{K}$ in double-CRT form.
Output: $\mathbf{c}' \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m(X^u), E')$
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 $D = \max(D_1, ..., D_d)$

Let η be the mapping $X \mapsto X^{u \bmod p}$
$(\hat{a}, \hat{b}) = (\eta(a), \eta(b)) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_{s(X^u)}(\Delta \cdot m(X^u), E)$
$\mathbf{c}' = \mathsf{KeySwt}(\hat{a}, \hat{b}, \mathbf{K}) \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(\Delta \cdot m(X^u), E')$
return $\mathbf{c}'$

E Detailed algorithm for ring packing

Algorithm 16: $\mathsf{PackLWE}$ Algorithm from [29]

Input: $[(\mathbf{a}^{(i)}, b^{(i)})]_{i<N} \in \mathsf{LWE}_{\mathbf{s}}^Q(\Delta \cdot m_i, E)$, packing key $\mathbf{K} := (\mathbf{a}, \mathbf{b}) \in \hat{\mathcal{R}}_Q^{n\ell \times 2}$ with E_R-subgaussian error.
Output: $(a, b) \in \hat{\mathcal{R}}_Q\mathsf{LWE}_z(\Delta \cdot m)$
Complexity: $O(n \cdot N \log(Q) \log(N))$ multiplications over $\mathbb{Z}_Q$
Noise growth: $(E, E_R) \mapsto O(\sqrt{N} \cdot E + \sqrt{n \cdot N \cdot \log(Q)} \cdot B \cdot E_{\mathcal{R}})$

for $0 \leq i < n$ **do**
  $\bar{a}_i := \sum_{j=0}^{N-1} a_i^{(j)} \cdot X^j$
$\bar{\mathbf{a}} = (\bar{a}_0, \ldots, \bar{a}_{n-1}) \in \hat{\mathcal{R}}_Q^n$
$\bar{b} = \sum_{i=0}^{N-1} b^{(i)} \cdot X^i$
$\mathbf{u} = g^{-1}(\bar{\mathbf{a}}) \in \hat{\mathcal{R}}_Q^{n \cdot L}$
$(a, b) = (-\mathbf{u} \cdot \mathbf{a}, \bar{b} - \mathbf{u} \cdot \mathbf{b}) \in \hat{\mathcal{R}}_Q^2$

F Proofs

F.1 Proof of Lemma 2, about shrinking matrices

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 $D_{k+1}, ..., D_d$ 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 $k+1 \leq i \leq d$, $Q_i = Q/D_i = (Q'/D_i) \cdot D^{(k)} = Q'_i \cdot D^{(k)}$, thus all entries are divisible by $D^{(k)}$.

Also, $\hat{Q}_i = [(Q/D_i)^{-1}]_{D_i} = [(Q'/D_i)^{-1} \cdot (D^{(k)})^{-1}]_{D_i} = \hat{Q}'_i \cdot (D^{(k)})^{-1} \bmod D_i$. 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 $\mathbf{y} := \mathsf{CRT}^{-1}(a,b)$. To show that $\mathbf{y} \cdot \mathbf{G}' = \alpha' \cdot (a,b) \mod Q'$, notice that the product by the first column of $\mathbf{G}'$, modulo Q', gives us

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

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

F.2 Lemma and proof of correctness of MsgExtract

Lemma 11 (Correctness of $\mathsf{MsgExtract}$**).** *Let the input ciphertext be* $\tilde{\mathbf{C}} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}^d_s(\alpha \cdot X^{\Delta \cdot m + e}, E)$*, 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* $f : \mathbb{Z}_t \to \mathbb{Z}_t$*, define the "test polynomial"* $t(X) \in \tilde{\mathcal{R}}$ *as*

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

Then, Algorithm 10, $\mathsf{MsgExtract}$*, outputs an LWE ciphertext* $\mathsf{LWE}^p_{\mathbf{s}}(\lfloor Q/t \rceil \cdot f(m), E') \in \mathbb{Z}_Q^{p+1}$*, where* $E' = O(\sqrt{dp} \cdot D \cdot E)$*.*

Proof. The correctness follows from the standard observation that 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 [13,8]. 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 3, but with the number of NTTs divided by two, since the first polynomial in the external product is zero. $\square$

F.3 Lemma about GSW key switching with less noise growth

Lemma 12 (Correctness and cost of GSW key switching with parallel reconstruction). *On input* $\mathbf{C} \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_z^d(\alpha \cdot m, E)$, $\mathbf{K}_z \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(z, E_k)$, $\mathbf{K}_s \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s, E_z)$, *and* $\mathbf{K}_{sz} \in \tilde{\mathcal{R}}_Q\mathsf{KS}_s^d(s \cdot z, E_s)$, *Algorithm 3 outputs* $\mathbf{C}' \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot m, E')$ *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 key switching for R/CLWE samples, Algorithm 14. 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 $\mathbf{K}_{sz} := [\mathbf{a}_k \mid \mathbf{b}_k := \mathbf{a}_k \cdot s + \mathbf{e}_k + s \cdot z \cdot \mathbf{g}] \in \tilde{\mathcal{R}}_Q^{d \times 2}$, where $\mathbf{g} \in \mathbb{Z}^d$ is the CRT gadget vector. Also, write $(a, b) = \mathtt{row}_i(\mathbf{C})$ with $b = a \cdot z + e + \Delta_i \cdot m$.

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

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

By defining $D = \max(D_1, ..., D_d)$, 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, $(a', b') \in \tilde{\mathcal{R}}_Q\mathsf{LWE}_s(-\Delta_i \cdot m \cdot s, E')$.

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 $\mathbb{Z}_{q_i}$. $\square$

F.4 Proof of Lemma 7, about noise growth of Algorithm 8

Proof. Algorithm 8 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 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^ρ. In this step, each a_k is defined as the output of Algorithm 7 on ciphertexts $\mathbf{C}_i \in \tilde{\mathcal{R}}_Q\mathsf{GSW}_s^d(\alpha \cdot X^{f_i}, E)$. Thus, by the noise analysis done in Algorithm 7, 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 7 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

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

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

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

□

G 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 8. Firstly, let's consider that no ciphertext shrinking is used, so d and ℓ 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 $\mathsf{EvalScalarProd}$, Algorithm 7, 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 ρ 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^ρ. 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 ρ, we have different dimensions and number of primes, d_i and ℓ_i for $0 \leq i \leq \rho$, where ℓ_ρ is the number of primes that we have in the very beginning (thus, $d_\rho > d_{\rho-1} > ... > d_0$ and $\ell_\rho > \ell_{\rho-1} > ... > \ell_0$).

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}, ℓ_{i+1}) to (d_i, ℓ_i) the n ciphertexts output by the m recursive calls.

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

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

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

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

H Further Improvements

In this section, we discuss some of the techniques that could be used to improve our construction or implementation. We note that, although we estimate the impact for some of them using experimental data, we do not present these techniques in our implementation. Their implementation is left as future work.

Optimal Parameter Selection We choose parameters in Section 5.3 based on a manual search. However, optimizing parameters is generally a complex task, requiring extensive parameter search, application considerations, and testing [4]. As such, a more extensive and formal parameter optimization could likely improve our results both for the amortized and non-amortized versions.

Memory usage The high memory requirements are certainly the main drawback of our construction, but there already are several ways of mitigating it. Table 5 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 recursive call to the INTT, 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 the next subsection.

Table 5: *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

Decomposed Automorphisms In [24], 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. It is hard to estimate the impact on the performance. In one hand, this technique could bring a slowdown of around 2 to 3 times, on the other, reducing drastically the memory usage reduces data movement, which speeds up the running times in practice. In [24], 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 [5], 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.