Fast amortized bootstrapping with small keys and polynomial noise overhead

Antonio Guimarães[1] and Hilder V. L. Pereira[2]

[1] IMDEA Software Institute, Madrid, Spain
[2] Institute of Computing, University of Campinas (UNICAMP), Brazil
antonio.guimaraes@imdea.org
hilder@unicamp.br

Abstract. Most homomorphic encryption (FHE) schemes exploit a technique called single-instruction multiple-data (SIMD) to process several messages in parallel. However, they base their security in somehow strong assumptions, such as the hardness of approximate lattice problems with *superpolynomial* approximation factor. On the other extreme of the spectrum, there are lightweight FHE schemes that have much faster bootstrapping but no SIMD capabilities. On the positive side, the security of these schemes is based on lattice problems with (low-degree) polynomial approximation factor only, which is a much weaker security assumption. Aiming the best of those two options, Micciancio and Sorrell (ICALP'18) proposed a new *amortized* bootstrapping that can process many messages at once, yielding sublinear time complexity per message, and allowing one to construct FHE based on lattice problems with polynomial approximation factor.

Some subsequent works on this line achieve near-optimal asymptotic performance, nevertheless, concrete efficiency remains mostly an open problem. The only existing implementation to date (GPV23, Asiacrypt 2023) requires keys of up to a hundred gigabytes while only providing gains for relatively large messages. In this paper, we introduce a new method for amortized bootstrapping where the number of homomorphic operations required per message is $O(h)$ and the noise overhead is $O(\sqrt{h\lambda}\log\lambda)$, where h is the Hamming weight of the LWE secret key and λ is the security parameter. This allows us to use much smaller parameters and to obtain faster running time. Our method is based on a new efficient homomorphic evaluation of sparse polynomial multiplication. We bootstrap 2 to 8-bit messages in 1.1 ms to 26.5 ms, respectively. Compared to TFHE-rs, this represents a performance improvement of 3.9 to 41.5 times while requiring bootstrapping keys up to 50.4 times smaller.

1 Introduction

Fully homomorphic encryption (FHE) [Gen09] is a powerful cryptographic primitive that allows one to evaluate arbitrary functions on encrypted data. However, evaluating a function homomorphically is typically orders of magnitude slower than evaluating it in clear, especially due to the *bootstrapping*, which is by far

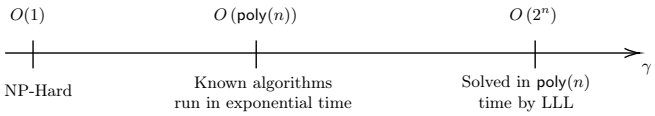

Fig. 1. Hardness of approximate lattice problems, such as GapSVP_γ, as the approximation factor γ grows as a function of the lattice dimension, n.

the most expensive operation in any FHE scheme. Thus, aiming to reduce the overhead of the homomorphic evaluation, many of the FHE schemes [BGV14, CKKS17, BBL17] can pack several messages into one ciphertext in such a way that each homomorphic operation operates on all the messages in parallel. This is known as single-instruction multiple data (SIMD) [SV14]. In particular, with those schemes, by evaluating one single bootstrapping, one refreshes many messages, so that the amortized time complexity (number of operations required to evaluate the boostrapping divided by the number of messages) becomes more acceptable.

The most popular FHE schemes are constructed on top of the learning with error (LWE) [Reg05] and ring learning with error (RLWE) [LPR10] problems, which are appealing because of the well-known worst-case to average-case reductions that allows us to base the security on lattice problems that where already believed to be hard even before the advent of homomorphic encryption. Roughly speaking, if one is able to solve in polynomial time LWE in dimension n, with modulus q, and discrete Gaussian errors with parameter σ, then one can solve hard problems like approximate shortest vector problem, GapSVP_γ, in dimension n and with approximation factor $\gamma = \tilde{O}(n \cdot q/\sigma)$ [Reg05, BLP+13].

However, the noise overhead, that is, the noise introduced by the bootstrapping, or, in other words, the noise "inside" the ciphertexts output by the bootstrapping, is superpolynomial for those schemes. Basically, if the bootstrapping outputs ciphertexts with noise magnitude e, then one has to choose the ciphertext modulus $q > e$, thus, bootstrapping with larger noise overhead implies larger approximation factor γ for the underlying lattice problem basing the securing of scheme. However, as we increase γ, solving the approximate lattice problems, as GapSVP_γ, becomes easier, therefore, supposing that GapSVP_γ is hard in the worst-case becomes a stronger assumption as γ grows, as illustrated in Figure 1. Ideally, we would like to construct our schemes under weak assumptions. In summary, for GBV and CKKS, since the bootstrapping noise overhead is superpolynomial, q has to be superpolynomial, and thus, the approximation factor γ of the lattice problems is also superpolynomial (in the security parameter). When one constructs FHE based on NTRU instead of (R)LWE, this noise overhead is even more problematic, since there are already known algorithms to efficiently solve the NTRU problem with large q [ABD16, KF17, DvW21].

Hence, it is important to construct FHE with bootstrapping noise overhead just polynomially large (in the security parameter). To achieve this, Alperin-Sheriff and Peikert [AP14] used the GSW scheme [GSW13] to bootstrap a very

simple LWE-based scheme which could only encrypt one bit. This line of research was further improved by the schemes FHEW [DM15] and TFHE [CGGI16], which, for the first time, were able to run the bootstrapping in less than one second. The time complexity of those bootstrapping algorithms, measured in number of GSW homomorphic operations, is $\tilde{O}(\lambda)$, that is, quasilinear in the security parameter. Moreover, the noise overhead is just a low-degree polynomial. On the negative side, they can only refresh one message per bootstrapping, which means that, when evaluating a function homomorphically, one has to run the bootstrapping several times, and this can end up being slow.

Thereupon, Micciancio and Sorrell proposed the concept of *amortized bootstrapping* [MS18], which consists in refreshing $O(\lambda)$ messages at once, but faster than running FHEW/TFHE bootstrapping $O(\lambda)$ times and still having $\mathsf{poly}(\lambda)$ noise overhead[3]. Subsequent works improved those results, reducing even further the amortized time complexity while maintaining polynomial noise overhead [GPvL23,DKMS24,LW23a,LW23b], however, they are not yet practical. In particular, to date, the only available implementation of this family of amortized bootstrapping [GPvL23] requires huge bootstrapping keys of up to a hundred gigabytes.

In this work, we use LWE and RLWE with sparse secrets to construct a practical amortized bootstrapping whose number of GSW homomorphic operations per message is linear in the Hamming weight of the RLWE secret key, and has much smaller bootstrapping key when compared to [GPvL23], while having quasilinear noise overhead. Moreover, we provide an open-source public implementation showing that our amortized running times are up to 32 times faster than [GPvL23] and up to 41 times faster than TFHE-rs. These results are summarized in Table 1.

1.1 Overview of existing amortized bootstrapping algorithms

The sequence of works [AP14, DM15, CGGI16] introduced a shift in the bootstrapping paradigm existing at the time, from slow bootstrapping algorithms refreshing many messages at once and producing refreshed ciphertext with superpolynomial noise to much simpler and faster bootstrapping refreshing a single message. The general framework is organized in two layers: one is the LWE-based scheme whose ciphertexts must be refreshed and the other is the (ring) GSW scheme used to homomorphically evaluate the decryption circuit of the first scheme, that is, to bootstrap it.

Given an LWE ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$ encrypting some message m, the first step of the decryption circuit consists in computing the inner product between $\mathbf{a}$ and the secret $\mathbf{s} \in \mathbb{Z}^n$, then subtracting it from b. Thus, the bootstrapping proceeds in a loop where at each iteration, the GSW scheme is used to generate an RLWE encryption of $X^{-a_i \cdot s_i}$, which is then accumulated so that at the end of the loop, one obtains an encryption of $X^{b-\mathbf{a}\cdot\mathbf{s}} = X^{e+m}$, where e is the noise in the

[3] Notice that [LW23c] does not satisfy this definition, because its noise overhead is $\Theta\left(\lambda^{\log(\lambda)}\right)$, which is superpolynomial.

Table 1: Functional bootstrapping with polynomial noise.

	Number of messages	Precision (bits)	Bootstrapping Key Size	Time	
				Total	Amortized
TFHE-rs [Zam22]	1	2	76.27MB[a]	9.8ms	9.8ms
		4	52.1MB	13.6ms	13.6ms
		6	488.5MB	131.1ms	131.1ms
		8	3.25GB	1.1s	1.1s
[GPvL23]	1024	7	63GB	14m30s	851ms
This work	2048	2	10.2MB	5.1s	2.5ms
		4	10.5MB	5.2s	2.5ms
		6	14MB	6.8s	3.3ms
		8	66MB	54.2s	26.5ms

[a] Could be reduced to 12.7MB with seeded-LWE. All others parameters could only be compressed up to a factor of 2
: Failure probability $\approx 2^{-64}$ and $\lambda \approx 128$ for all cases. Execution time includes all necessary key switchings. See Section 7.2 for key switching key sizes of each implementation.

input ciphertext. In other words, the linear part of the decryption is evaluated "in the exponent" of the indeterminate X. Finally, an extraction procedure maps this RLWE encryption of X^{e+m} to an LWE encryption m. It is evident that this approach requires $\tilde{O}(n) = \tilde{O}(\lambda)$ GSW homomorphic operations per message.

Then, Micciancio and Sorrel [MS18] proposed a packing algorithm to combine $O(\lambda)$ LWE ciphertexts encrypting messages m_i's into one single RLWE ciphertext encrypting all the $O(\lambda)$ messages and to follow the LWE-GSW approach, but with a RLWE ciphertext as input. However, the inner-product in the LWE decryption circuit is now a polynomial product, as RLWE samples are composed by pairs of polynomials. Thus, [MS18] proposed an algorithm to homomorphically evaluate a DFT-like algorithm using GSW in such a way that, at the end, they obtained several ciphertexts encrypting messages of the form $X^{e_i+m_i}$, from which the extraction procedure could be applied and refreshed LWE ciphertexts encrypting the m_i's could be produced. Their amortized bootstrapping has noise overhead $\tilde{O}(\lambda^{2+3\cdot\rho})$ and requires $\tilde{O}(3^\rho \cdot \lambda^{1+1/\rho})$ GSW homomorphic operations, where ρ is a parameter controlling the recursion depth of the DFT-like algorithm. Since it refreshes $O(\lambda)$ messages at once, it only requires $\tilde{O}(3^\rho \cdot \lambda^{1/\rho})$ GSW homomorphic operations per message, i.e., it has sublinear amortized time complexity.

This approach was further improved [GPvL23, DKMS24, LW23a, LW23b], still using the GSW scheme to evaluate some DFT-like algorithm, and the asymptotic cost per message is now only $O(\mathsf{poly}(\log \lambda))$. However, those amortized bootstrapping are still far from practical, as only one of them was actually implemented [GPvL23] and ended up needing gigabytes of bootstrap-

ping keys. The main limitation of this approach is the requirement for increased parameters to instantiate the GSW scheme, which brings a huge overhead to each homomorphic operation and to size of the bootstrapping key, as it is actually a list of GSW ciphertexts. Also, since performing computation in the exponent is implemented mainly with homomorphic multiplication (*e.g.*, $\mathsf{Enc}(X^{m_1}) \cdot \mathsf{Enc}(X^{m_2}) = \mathsf{Enc}(X^{m_1+m_2})$), we would like to use efficient homomorphic multiplication, such as the external product [CGGI16, BIP+22]. However, because of their multiplicative depth, the DFT-like algorithms force us to use regular GSW multiplication, which means a logarithmic slowdown in practice. Instead, in our approach, we are able to just use external products and small parameters, therefore, having fast running time and small bootstrapping keys in practice.

1.2 Overview of our contributions and techniques

New GSW-friendly sparse-by-dense polynomial multiplication. Firstly, we propose a new algorithm that allows us to multiply, "in the exponent", two polynomials a and s, where a is in clear, s is encrypted, has binary coefficients, and has Hamming weight[4] h. This is a key piece of many bootstrapping algorithms, since it corresponds to evaluating the first part of the decryption circuit of RLWE-based schemes.

Let $\mathsf{idx}(s)$ be the set containing the degrees of nonzero coefficients of s and let $\odot$ represent a monomial-by-polynomial multiplication. Then, one can compute $b - a \cdot s$ as the summation of h monomial-by-polynomial multiplications:

$$b - a \cdot s = b - a \cdot \sum_{i \in \mathsf{idx}(s)} X^i = b - \sum_{j \in \mathsf{idx}(s)} \left(\sum_{i=0}^{N} a_i X^i \right) \odot X^j \tag{1}$$

It is clear that above equation can be computed in $O(hN)$ operations on cleartext. However, when the indexes j are encrypted, the situation is less clear. Considering this, we introduce the algorithm MPmul (Algorithm 1 in Section 3), for homomorphically evaluating monomial-by-polynomial multiplication (with messages in the exponent) in time $O(N \log B)$, where B is a bound for the degree of the monomial. This algorithm only requires external products and Galois Automorphisms and has noise overhead $O(\log B)$.

We also generalize this algorithm so that it can handle encrypted polynomials s with non-binary coefficients, that is, polynomials s with Hamming weight h and coefficients in $\{-\rho, ..., -1, 0, 1, ..., \rho\}$. This allows us to bootstrapping FHE schemes that use more general secret keys instead of simply binary ones.

[4] We define the Hamming weight of a polynomial as the number of nonzero coefficients it has.

New decryption circuit compatible with our MPMUL algorithm It is clear that Equation 1 could be evaluated homomorphically as

$$b - a \cdot s = b - \sum_{j \in \mathsf{idx}(s)} \text{MPMUL}(a, \mathsf{Enc}(X^j)) \tag{2}$$

Considering, for simplicity, $B = N$, this would result in an amortized bootstrapping with complexity $O(hN \log N)$ homomorphic operations (i.e., $O(h \log N)$ per message). However, since all the values are encrypted "in the exponent", the summation would actually require homomorphic multiplications, therefore, we would either have to use expensive RLWE-times-RLWE products or use GSW multiplication in our MPMUL instead of external products so that we could perform another level of multiplications the evaluate the sum. Moreover, this would increase a lot the parameters used in practice, probably giving us parameters slightly better than [GPvL23] but still worse than [CGGI16] and [LMK+23]. To avoid this, we proceed as follows: Let $\tilde{j} = \mathsf{idx}(s)$ be the set of exponents of the nonzero coefficients of s. Checking again Equation 1, we see that $b - \sum_{j \in \mathsf{idx}(s)} a \odot X^j$ can be rewritten as

$$\left(\left(\left(b \odot X^{-\tilde{j}_0} - a\right) \odot X^{\tilde{j}_0 - \tilde{j}_1} - a\right) \odot X^{\tilde{j}_1 - \tilde{j}_2} \cdots - a\right) \odot X^{\tilde{j}_h}$$

Now, using MPMUL, we have:

$$\text{MPMUL}\left(\ldots \text{MPMUL}\left(\text{MPMUL}\left(b, X^{-\tilde{j}_0}\right) - a, X^{\tilde{j}_0 - \tilde{j}_1}\right) \cdots - a, X^{\tilde{j}_h}\right)$$

With this formulation, we no longer have to use GSW multiplications, and we can evaluate the bootstrapping entirely with external products, which are faster by a factor of $\Theta(\log \lambda)$. In more detail, notice that all uses of MPMUL are now nested with at least one operand ($X^{\tilde{j}}$) being a fresh encryption, which can be then defined as the bootstrapping key. Moreover, all subtractions, although still implemented as multiplications, have at least one operand (the polynomial a) as a cleartext, thus not adding any noise nor requiring expensive ciphertext-ciphertext multiplication. Hence, we are able to evaluate the bootstrapping with $O(hN \log N)$ external products. Moreover, noise and key size are only $O(h \log N)$, enabling us to have use the same (or better) parameters as [CGGI16] and [LMK+23] while having smaller evaluation keys.

As a last adjustment to this base technique, notice that we are now always multiplying by a monomial with degree given by the difference of consecutive elements from $(\tilde{j}_0, ..., \tilde{j}_{h-1}) = \mathsf{idx}(s)$ and it is easy to establish a high-probability bound $B = O(N/h)$ for the values $\tilde{j}_i - \tilde{j}_{i+1}$, allowing us to run MPMUL in time $O(N \log (N/h))$, reducing bootstrapping complexity to $O(hN \log (N/h))$ and both the noise overhead and the key size to $O(h \log (N/h))$. We provide further detail in Sections 3 to 6.

New packing algorithms exploiting low-Hamming weight secrets We propose a new algorithm to pack many LWE ciphertexts encrypting messages

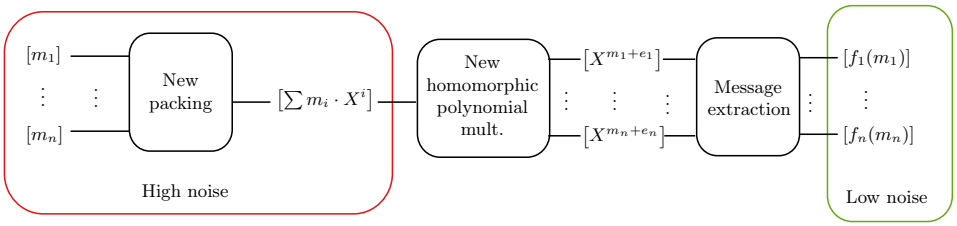

Fig. 2. Overview of our function amortized bootstrapping.

$m_i \in \mathbb{Z}$ under a secret key s with Hamming weight h into a single RLWE encryption of $m(X) = \sum m_i X^i$, which can then be used as input of our homomorphic decryption circuit. Our algorithm presents new time-noise tradeoffs, as it is possible to set its parameters so that it introduces more noise in the packed ciphertexts but runs faster.

Putting it all together, we are able to evaluate the bootstrapping over $O(\lambda)$ messages at once, as illustrated in Figure 2. The process starts with high-noise LWE ciphertexts encrypting messages $m_i \in \mathbb{Z}_t$ and finishes with low-noise LWE ciphertexts encrypting $f_i(m_i)$, where f_i are arbitrary functions from $\mathbb{Z}_t$ to $\mathbb{Z}_t$.

Proof-of-concept implementation and practical results We provide an open-source[5] C implementation of our amortized functional bootstrapping algorithm for many message spaces. We compare our results to [GPvL23], the only public implementation of amortized bootstrapping to date, and to TFHE-rs [Zam22], the fastest implementation of non-amortized bootstrapping we are aware of. The practical results are summarized in Table 1. Firstly, we notice that our bootstrapping keys are much smaller, being 3 orders of magnitude less than the ones of [GPvL23] and even about 2 orders of magnitude less than TFHE-rs for some message spaces. Moreover, we obtained speedups of 32 times over [GPvL23] and up to 41 times for TFHE-rs.

2 Preliminaries

For any vector $\mathbf{u}$, we denote the infinity norm by $\|\mathbf{u}\|$ and the Euclidean norm by $\|\mathbf{u}\|_2$. The Hamming Weight of a vector $\mathbf{u}$, represented by $\mathsf{HamWeight}(\mathbf{u})$, is the number of nonzero entries u_i from $\mathbf{u}$. For any polynomial $a = \sum_{i=0}^{d} a_i \cdot X^i$, we define the norm (and the Hamming weight) of a as the norm (and the Hamming weight) 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 (with degree less than the degree of f), and thus the norm (and the Hamming weight) of a is simply the norm (and the Hamming weight) of a'.

[5] Github: TO BE PUBLISHED WITH THE PAPER.

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 $\mathcal{R}$. For any positive integer Q, we define $\mathcal{R}_Q := \mathcal{R}/(Q\mathcal{R})$ i.e., the same ring as before but with coefficients of the elements reduced modulo Q.

Probability distributions We denote by $\mathcal{U}(X)$ the uniform distribution over a finite set X and by $x \leftarrow D$ we mean that s was sampled following the distribution D. The discrete Gaussian distribution with parameter σ and centered on c samples $z \in \mathbb{Z}$ with probability $\rho(z)/(\sum_{y\in\mathbb{Z}} \rho(y))$, where $\rho(x) = \exp(-\pi(x - c)^2/\sigma^2)$. A random variable X is subgaussian with parameter $\sigma > 0$, in short σ-subgaussian, if for all $t \in \mathbb{R}$ it holds that $\mathbb{E}[\exp(2\pi tX)] \leq \exp(\pi\sigma^2 t^2)$. In this case, we also write $X \sim SG(E)$. The "tails" os subgaussians are bounded by a Gaussian function of standard deviation σ, that is $\forall b \in \mathbb{R},\ \Pr[|X| \geq b] \leq 2\exp(-\pi b^2/\sigma^2)$. From this, one can bound the absolute value of X with overwhelming probability, since fixing $b = \sigma\sqrt{\lambda/\pi}$, implies $\Pr[|X| \geq \sigma\sqrt{\lambda/\pi}] \leq 2\exp(-\pi(s\sqrt{\lambda/\pi})^2/s^2) = 2\exp(-\lambda) < 2^{-\lambda}$. In other words, $|X|$ is less than $\sigma\sqrt{\lambda/\pi}$ with probability at least $1 - 2^{-\lambda}$. Linear combinations of independently distributed subgaussians are again subgaussians, i.e., given independent σ_i-subgaussian distributions X_i's, then for any $\mathbf{u} = (u_1, ..., u_n) \in \mathbb{R}^n$, it holds that $Z := \sum_{i=1}^n u_i X_i$ is σ-subgaussian with $\sigma = \sqrt{\sum_{i=1}^n c_i^2\sigma_i^2}$. In particular, if all σ_i are equal to some $\hat{\sigma}$, then, $\sigma = \hat{\sigma} \cdot \|\mathbf{c}\|_2$. Also, since $\|\mathbf{c}\|_2 \leq \sqrt{n} \cdot \|\mathbf{c}\|_\infty$, we have $\sigma\hat{\sigma} \cdot \sqrt{n} \cdot \|\mathbf{c}\|_\infty$. 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 two polynomials a and b of degree less than n following subgaussians with parameters σ_a and σ_b, we can see that for $c = a \cdot b \bmod f$, each coefficient c_i is $(\sqrt{n} \cdot \sigma_a \cdot \sigma_b)$-subgaussian, but the c_i's are not necessarily independent, thus, to say that c is $(\sqrt{n} \cdot \sigma_a \cdot \sigma_b)$-subgaussian, we use the independence heuristic, which is commonly to simplify analysis of FHE schemes [CGGI16, CGGI17, BIP+22].

Independence heuristic All the coefficients of the noise terms of the RLWE samples appearing in the linear combinations we consider are assumed to be independent and concentrated. In more detail, they follow σ-subgaussian, where σ^2 is their variance.

Variants of LWE The learning with errors problem [Reg05] with parameters n, q, and σ, denoted $\mathsf{LWE}_{n,m,q,\sigma}$, an attacker has to find a secret vector $\mathbf{s} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ given up to m samples of the form $(\mathbf{a}_i, b_i)$, where $\mathbf{a}_i$ is uniform in $\mathbb{Z}_q^n$ and $b_i := \mathbf{a}_i \cdot \mathbf{s} + e_i \bmod q$, with e_i following a discrete Gaussian distribution with parameter σ.

In the ring version of LWE, known as $\mathsf{RLWE}_{N,m,q,\sigma}$ [LPR10], we fix the ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and a secret polynomial $s \leftarrow \mathcal{U}(\mathcal{R}_q)$. Then, an attacker is given up to m samples of the form (a_i, b_i), where a_i is uniformly sampled from $\mathcal{R}_q$ and $b_i := a_i \cdot s + e_i \in \mathcal{R}_q$, for some small noise term e_i following a discrete Gaussian distribution with parameter σ, and the attacker has to find s.

Both RLWE and LWE problems also have decision versions, where the attacker has to distinguish the samples (a_i, b_i) from (a_i, u_i) where u_i is uniformly distributed. Moreover, both problems are often instantiated with a secret having a *known* Hamming weight. Following [CHK+17], we define $S_{n,h,r} := \{\mathbf{u} \in \mathbb{Z}^n : \mathsf{HamWeight}(\mathbf{u}) = h \text{ and } \|\mathbf{u}\|_\infty \leq r\}$, i.e., the set of vectors with Hamming weight h and entries in the interval $[\![-r, r]\!]$, and we use $\mathsf{spLWE}_{n,m,q,\sigma,r,h}$ to denote $\mathsf{LWE}_{n,m,q,\sigma}$, but with secret $\mathbf{s}$ sampled uniformly from $S_{n,h,r}$ instead of $\mathbb{Z}_q^n$. Analogously, we define $\mathsf{spRLWE}_{n,m,q,\sigma,r,h}$ as the sparse-secret RLWE (that is, with secret polynomial s having its coefficient vector sampled from $S_{N,h,r}$).

2.1 LWE and RLWE encryption schemes

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

$$\mathsf{LWE}_{\mathbf{s}}^q(\Delta m, E) = \{(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1} : b = [\mathbf{as} + e + \Delta \cdot m]_q \text{ and } e \sim SG(E)\}$$

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

$$\mathcal{R}_q\mathsf{LWE}_s(\Delta m, E) := \{(a, b) \in \mathcal{R}_q^2 : b = [as + e + \Delta \cdot m]_q \text{ and } e \sim SG(E)\}$$

In both cases, 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 outputting $\lfloor e'/\Delta \rceil \bmod t$. If $\|e + \Delta \cdot m\| < q/2$ and $\|e\| < q/(2t)$, then the decryption correctly outputs $m \bmod t$.

We also briefly describe the ring variant of the GSW scheme [GSW13] presented in [DM15]. By fixing a decomposition base β and a gadget vector $\mathfrak{g} = (\beta^0, \beta^1, ..., \beta^{\ell-1})$, where $\ell := \lfloor \log_\beta(q) \rfloor + 1$, the set of GSW ciphertexts encrypting a message $m \in \mathcal{R}$, under a secret key s, and with E-subgaussian noise is

$$\mathcal{R}_q\mathsf{GSW}_s^\ell(m, E) := \{[\mathbf{a}, \mathbf{a} \cdot s + \mathbf{e}] + m \cdot \mathfrak{g} \otimes \mathbf{I}_2 \in \mathcal{R}_q^{2\ell \times 2} : \mathbf{e} \sim SG(E)\}$$

At last, we also define the set of key-switching keys from a key s to a key z, with E-subgaussian noise. For this, consider a decomposition base β_{ksk} and a modulus q, then define $\ell_{\mathsf{ksk}} := \lfloor \log_{\beta_{\mathsf{ksk}}}(q) \rfloor + 1$ and $\mathfrak{g}_{\mathsf{ksk}} = (\beta_{\mathsf{ksk}}^0, \beta_{\mathsf{ksk}}^1, ..., \beta_{\mathsf{ksk}}^{\ell-1})$. Then, this set is defined as

$$\mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(s \to z, E) := \{[\mathbf{a}, \mathbf{a} \cdot z + \mathbf{e} + s \cdot \mathfrak{g}_{\mathsf{ksk}}] \in \mathcal{R}_q^{\ell_{\mathsf{ksk}} \times 2} : e \sim SG(E)\}$$

Common homomorphic operations In this section, we describe in a high-level the homomorphic operations we will use in our boostrapping algorithms.

- $\mathsf{FHE.Add}$: homomorphically add two ciphertexts of the same type, e.g., maps $(\mathbf{c}_0, \mathbf{c}_1) \in \mathcal{R}_q\mathsf{LWE}_s(m_0, E_0) \times \mathcal{R}_q\mathsf{LWE}_s(m_1, E_1)$ to $\mathbf{c} \in \mathcal{R}_q\mathsf{LWE}_s(m_0 + m_1, E)$ where $E \leq \sqrt{E_0^2 + E_1^2}$.

- $\mathsf{FHE.ModSwt}$: Given $\hat{c} \in \mathsf{LWE}^{Q}_{\mathbf{s}}(\lfloor Q/t \rceil \cdot m, \hat{E})$ and $q \in \mathbb{N}$, output $c \in \mathsf{LWE}^{q}_{\mathbf{s}}(\lfloor q/p \rceil \cdot m, E)$, with $E^2 \leq (\hat{E} \cdot (q/Q))^2 + (\|\mathbf{s}\|_2 / 2)^2$.
- $\mathsf{FHE.KeySwt}$: Given $\hat{c} \in \mathsf{LWE}^{q}_{\mathbf{s}}(m, \hat{E})$, and a key-switching key ksk from $\mathbf{s} \in \mathbb{Z}^N$ to $\mathbf{z} \in \mathbb{Z}^n$, output $c \in \mathsf{LWE}^{q}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E)$, with $E^2 \leq \hat{E}^2 + n \cdot \log_{\beta_{\mathsf{ksk}}}(q) \cdot \beta^2_{\mathsf{ksk}} \cdot E^2_{\mathsf{ksk}}$, where β_{ksk} is the decomposition base of ksk. There are also key-switching procedures for RLWE ciphertexts.
- $\mathsf{FHE.ExtProd}$: given $\mathbf{c}_0 \in \mathcal{R}_q\mathsf{LWE}_s(\Delta \cdot m_0, E_0)$ and $\mathbf{C}_1 \in \mathcal{R}_q\mathsf{GSW}^{\ell}_s(m_1, E_1)$, it outputs $\mathbf{c} \in \mathcal{R}_q\mathsf{LWE}_s(\Delta \cdot m_0 \cdot m_1, E)$ where $E \leq \sqrt{\ell N \cdot \beta^2 \cdot E_1^2 + \|m_1\|_2^2 \cdot E_0^2}$, where β is the decomposition base.
  Although the external product multiplies the messages, it is often use to add them "in the exponent", that is, instead of encrypting $m_i \in \mathbb{Z}$, one encrypts X^{m_i}. Then, external products produces encryptions of $X^{m_i + m_j}$.
- $\mathsf{FHE.Automorphism}$: Given $\hat{c} \in \mathcal{R}_q\mathsf{LWE}_{\mathbf{s}}(m, \hat{E})$, odd integer $k \in [\![1, 2N]\!]$, and $\mathsf{ksk} \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(s \to s(X^k), E_{\mathsf{ksk}})$, output $c \in \mathcal{R}_q\mathsf{LWE}_{\mathbf{s}}(m(X^k), E)$, with $E^2 \leq \hat{E}^2 + N \cdot \log_{\beta_{\mathsf{ksk}}} q \cdot \beta^2_{\mathsf{ksk}} \cdot E^2_{\mathsf{ksk}}$.

2.2 Test vectors and extraction

Now, we describe the well-known technique to "extract" the message from the exponent of X to constant coefficient [CGGI16, CIM19]. The RLWE sample we obtain at the end of the main loop of the bootstrapping, which encrypts something of the form $X^{e + \lfloor 2N/p \rceil m}$, can be converted to an RLWE sample encrypting a polynomial $u(X)$ whose coefficient u_0 is equal $f(m)$, for any function $f : \mathbb{N}_{p/2} \to \mathbb{N}_{p/2}$. To do so, we define the polynomial $t(X) = -\sum_{i=0}^{N-1} f(\lfloor \frac{ip}{2N} \rfloor) X^{N-i}$, then, $u = t(X) \cdot X^{e + \lfloor 2N/p \rceil m} \bmod X^N + 1$ has the desired property, $u_0 = f(m)$. If f is public, we then define its corresponding test vector as $\mathbf{t} := (0, \Delta \cdot t(X)) \in \mathcal{R}^2$, which is a trivial and noiseless RLWE ciphertext. Otherwise, we actually encrypt $t(X)$, obtaining $\mathbf{t} = \mathcal{R}_q\mathsf{LWE}_s(\Delta \cdot t(X))$.

Moreover, given $\mathbf{c} = \mathcal{R}_q\mathsf{LWE}_s(\Delta \cdot u, E)$, one can extract $\hat{\mathbf{c}} \in \mathsf{LWE}^{q}_{\mathbf{s}}(\Delta \cdot u_0, E)$ for free, thus, obtaining an LWE encryption of m.

3 Homomorphic multiplication of monomial and polynomial

In this section, we present an algorithm that homomorphically performs the multiplication $u \cdot Y^v \bmod Y^n + 1$, where u is an arbitrary polynomial in $\mathbb{Z}[Y]/\langle Y^n + 1 \rangle$, v is an integer between 0 and n, and both are encrypted. Notice that multiplying by Y^v means that we want to rotate the coefficients of u by v and multiply v of them by -1, as follows

$$(u_0, a_1, ..., u_{n-1}) \mapsto (\underbrace{-u_{n-v}, -u_{n-v+1}, ..., -u_{n-1}}_{\text{negate } v \text{ positions}}, u_0, u_1, ..., u_{n-b-1})$$

By starting with a list of ciphertext $c_i \in \mathcal{R}_q\mathsf{LWE}_s(X^{u_i})$ encrypting the coefficients of u (in the exponent), we could simply rotate the list of c_i's itself and

multiply some of them, however, since we do not know the value of b, performing this rotation in not trivial.

Our solution is as follows. Let's say we can homomorphically multiply by $Y^{v_i 2^i}$ where $v_i \in \{0,1\}$, then we can multiply by Y^v by sequentially multiplying by $Y^{v_0 2^0}, Y^{v_1 2^1}, ...,$ and $Y^{v_L 2^L}$, where $(v_0, v_1, ..., v_L)$ is the binary decomposition of v. But multiplying by $Y^{v_i 2^i}$ is the same as evaluating the following function:

$$f(u_0, ..., u_{n-1}) = \begin{cases} (u_0, u_1, ..., u_{n-1}) & \text{if } v_i = 0 \\ (-u_{n-2^i}, ..., -u_{n-1}, u_0, ..., u_{n-2^i-1}) & \text{if } v_i = 1 \end{cases}$$

Now, notice that the number of elements that have to be multiplied by -1 is 2^i, where i is known, in other words, we know exactly which elements have to negated (the first 2^i) and which only have to be rotated. Also, since the homomorphic CMUX [CGGI16] allows us to perform two different computations depending on an encrypted bit, we can differentiate the two scenarios. In more detail, given $\mathbf{C}_i \in \mathcal{R}_q\mathsf{GSW}_s^\ell(v_i)$, then for $2^i \leq j \leq n - 2^i - 1$, it is enough to evaluate

$$\hat{c}_j \leftarrow \left(c_{j-2^i} - c_j\right) \boxdot \mathbf{C}_i + c_j.$$

One can see that if $v_i = 0$, then $\hat{c}_j = c_j$, i.e., no rotation is performed, and if $v_i = 1$, then $\hat{c}_j = c_{j-2^i}$, that is, a rotation by 2^i is performed, as expected. And for the first 2^i positions, we can use the same CMUX, but we first apply the automorphism $X \mapsto X^{-1}$ so that the value we obtain is either an encryption of X^{u_j} itself (when $b_i = 0$) or an encryption of $X^{-u_{n-2^i+j}}$, that is, both the rotation by 2^i and the negation happens. In more detail, for $0 \leq j \leq 2^i$, we can evaluate

$$\hat{c}_j \leftarrow \left(\mathsf{FHE.Automorphism}(c_{n-2^i+j}, -1) - c_j\right) \boxdot \mathbf{C}_i + c_j$$

This shows how we can obtain the list $(\hat{c}_0, ..., \hat{c}_{n-1})$ representing the encryption of $u \cdot Y^{v_i \cdot 2^i} \bmod Y^n + 1$. Hence, we just need to repeat this procedure for $0 \leq i \leq L$, where L is any upper bound on the number of bits of v (for instance, one could set $L = \lfloor \log n \rfloor + 1$). This gives us the full homomorphic multiplication, which is shown in detail in Algorithm 1, MPMUL.

To analyze the noise growth, we can focus on line 3, as it introduces more noise than line 5. Suppose that at the i-th iteration, we have a list of RLWE ciphertexts $(\mathbf{c}_0, ..., \mathbf{c}_{n-1})$ with subgaussian noise with parameter E_i, a GSW ciphertext with whose noise is E_G-subgaussian, and a key-switching key with E_k-subgaussian noise. Then, from the description of the external product and the automorphism in Section 2.1, after the automorphism we have $\tilde{E}$-subgaussian noise with $\tilde{E} \leq \sqrt{E_i^2 + N \cdot \log_{\beta_{\mathsf{ksk}}} q \cdot \beta_{\mathsf{ksk}}^2 \cdot E_k^2}$, where β_{ksk} is the decomposition base used to key switch. Thus, after the external product, we have E_{i+1}-subgaussian noise where $E_{i+1} \leq \sqrt{\ell N \beta^2 E_G^2 + |b_i| \tilde{E}^2} \leq \sqrt{\ell N \beta^2 E_G^2 + N \log_{\beta_{\mathsf{ksk}}}(q) \beta_{\mathsf{ksk}}^2 E_k^2 + E_i^2}$. Hence, after $\log B$ iterations, the output has noise with parameter $E_{out} \leq \sqrt{(\log B)\left(\ell N \beta^2 E_G^2 + N \log_{\beta_{\mathsf{ksk}}}(q) \beta_{\mathsf{ksk}}^2 E_k^2\right) + E_0^2}$. This gives us the following lemma.

Algorithm 1: Monomial-times-polynomial multiplication (MPMUL)

Input : a list $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_z(X^{u_i})$ for $0 \le i < n$ representing a polynomial $u(Y)$ encrypted in the exponent
Input : a list $\mathbf{C}_i \in \mathcal{R}_q\mathsf{GSW}^{\ell}_z(v_i)$ for $0 \le i < \log B$ where $v_i \in \{0,1\}$ and $\sum_{i=0}^{\log B-1} v_i 2^i = v < B$
Input : a key $\mathsf{ksk} \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to z, E_{\mathsf{ksk}})$ for the automorphism
Output: a list $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_z(X^{w_i})$ for $0 \le i < n$ representing an encryption of a polynomial $w(Y)$ such that $w = u \cdot Y^v \mod (Y^n+1)$

for $0 \le i \le \log B - 1$ **do**
  /* For each i, multiply by $Y^{v_i 2^i} \bmod Y^n+1$ */
  for $0 \le j < 2^i$ **do**
    $\hat{\mathbf{c}}_j \leftarrow \left(\mathsf{FHE.Automorphism}(\mathbf{c}_{n-2^i+j}, -1) - \mathbf{c}_j\right) \boxdot \mathbf{C}_i + \mathbf{c}_j$
  for $2^i \le j < n - 2^i$ **do**
    $\hat{\mathbf{c}}_j \leftarrow \left(\mathbf{c}_{j-2^i} - \mathbf{c}_j\right) \boxdot \mathbf{C}_i + \mathbf{c}_j$
  $(\mathbf{c}_0, ..., \mathbf{c}_{n-1}) \leftarrow (\hat{\mathbf{c}}_0, ..., \hat{\mathbf{c}}_{n-1})$
return $(\mathbf{c}_0, ..., \mathbf{c}_{n-1})$

Lemma 3.1 (Noise overhead of MPMUL). *Let* $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_z(X^{u_i}, E_0)$ *and* $\mathbf{C}_j \in \mathcal{R}_q\mathsf{GSW}^{\ell}_z(v_j, E_G)$ *be the input ciphertexts of Algorithm 1. Also, consider that input key-switching key is* $\mathsf{ksk} \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to z, E_{\mathsf{ksk}})$ *and let* $\ell_{\mathsf{ksk}} := \lfloor \log_{\beta_{\mathsf{ksk}}} q \rfloor + 1$. *Then, the output ciphertexts have* E_{out}*-subgaussian noise with*

$$E_{out} \le \sqrt{(\log B)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_0^2}$$

4 Homomorphically multiplying the secret key

In this section, we describe three different ways of using our algorithm MPMul to perform the main step of the decryption that has to be evaluated homomorphically during the bootstrapping. To avoid confusion, we use ring $\mathcal{R} := \mathbb{Z}[X]/\langle X^N+1\rangle$ for the bootstrapping key and the ring $\mathcal{R}_Y := \mathbb{Z}[Y]/\langle Y^n+1\rangle$ for the input ciphertext. Basically, the homomorphic computation is performed in the exponent of X, but we are evaluating functions on $\mathcal{R}_Y$.

Thus, consider that we have an RLWE ciphertext $(a(Y), b(Y) = a(Y)\cdot s(Y) + e(Y) + m(Y))$, defined modulo Y^n+1 and a bootstrapping key encrypting the h nonzero coefficients of $s(Y)$ in some way. Then, we would like to compute $p(Y) := b(Y) - a(Y)s(Y) \bmod Y^n+1$ in the exponent, that is, to obtain ciphertexts encrypting X^{p_i}. From this, we can use standard techniques to extract the message from the exponent, generating LWE samples.

Define $\tilde{j} = \mathsf{idx}(s)$ as the sequence of exponents of the nonzero coefficients of s. For instance, if $s(Y) = 1 - 1\cdot Y^5 + 1 \cdot Y^7$, then $\tilde{j} = (0, 5, 7)$. Also, define $\mathbf{d} := \mathsf{diff}(s) \in [\![-n, n]\!]^{h+1}$ as $\mathbf{d}[0] = -\tilde{j}_0$, $\mathbf{d}[h] = -\tilde{j}_{h-1}$, and $\mathbf{d}[i] = \tilde{j}_{i-1} - \tilde{j}_i$ for $1 \le i \le h-1$.

Algorithm 2: Subtract $a \cdot s$ when s is a binary sparse secret (BIN-SAB)

Input : an RLWE ciphertext $(a, b) \in \mathcal{R}_Y^2$ encrypting $m(Y) = \sum_{i=0}^{n} m_i Y^i$ under a key s with noise e. For $0 \leq i < n$, $a_i, b_i \in \mathbb{Z}_{2N}$
Input : GSW ciphertexts encrypting the binary decomposition of $\mathbf{d} := \mathsf{diff}(s)$, as required by MPMUL, i.e., $\mathbf{C}_{i,j} \in \mathcal{R}_q \mathsf{GSW}_z^{\ell}(d_{i,j})$ for $0 \leq i \leq h$, $0 \leq j < \log B$, where $\sum_{j=0}^{\log B - 1} d_{i,j} 2^j = \mathbf{d}[i]$
Input : a key $\mathsf{ksk} \in \mathcal{R}_q \mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to z, E_{\mathsf{ksk}})$ for the automorphism
Input : test vectors $\mathbf{t}_0, ..., \mathbf{t}_{n-1}$ encoding LUTs as polynomials $t_i(X)$
Output: a list $\mathbf{c}_i \in \mathcal{R}_q \mathsf{LWE}_z(t_i(X) \cdot X^{m_i + e_i})$ for $0 \leq i < n$.

Let $\mathbf{c} := (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$ where $\mathbf{c}_i = \mathsf{FHE.MultPtxt}(\mathbf{t}_i, X^{b_i})$
for $0 \leq i \leq h - 1$ **do**
  /* Multiply by $X^{\mathbf{d}[i]}$ */
  Let $\mathbf{C} = (\mathbf{C}_{i,1}, ..., \mathbf{C}_{i,\log n - 1})$
  $\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
  /* Subtract $a(Y)$ */
  for $0 \leq k \leq n - 1$ **do**
    $\mathbf{c}_k \leftarrow \mathsf{FHE.MultPtxt}(\mathbf{c}_i, X^{-a_k})$
  $\mathbf{c} \leftarrow (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$
Let $\mathbf{C} = (\mathbf{C}_{h,1}, ..., \mathbf{C}_{h,\log n - 1})$
$\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
return $\mathbf{c}$

4.1 Bootstrapping with binary secrets

Firstly, let's consider the simplest case where s has binary coefficients. Define $r := b - a \cdot s = b - \sum_{j \in \mathsf{idx}(s)} a \odot Y^j$. Then, we have

$$r = \left(\left(\left(b \odot Y^{\mathbf{d}[0]} - a\right) \odot Y^{\mathbf{d}[1]} - a\right) \odot Y^{\mathbf{d}[2]} \cdots - a\right) \odot Y^{\mathbf{d}[h]} \tag{3}$$

Now, using MPMUL, we have:

$$\text{MPMUL}\left(\ldots \text{MPMUL}\left(\text{MPMUL}\left(b, Y^{\mathbf{d}[0]}\right) - a, Y^{\mathbf{d}[1]}\right) \cdots - a, Y^{\mathbf{d}[h]}\right)$$

Notice that since a is a known polynomial, subtracting it means that we can simply multiply each RLWE ciphertext $\mathbf{c}_i$ returned by MPMUL by X^{-a_i}, which is almost for free and does not increase the noise of the ciphertexts. This gives us Algorithm 2, which takes as input the values $\mathbf{d}[i]$ decomposed into bits and encrypted as GSW ciphertexts, as required by MPMUL, and the RLWE ciphertext to be refreshed, i.e., $(a(Y), b(Y)) \in \mathcal{R}_Y^2$ such that $b = a \cdot s + e + m \bmod Y^n + 1$, then computes a list of RLWE ciphertexts encrypting $u := b - a \cdot s \bmod Y^n + 1$ in the exponent. As an optimization for the moment it is used in the functional bootstrapping, it also receives test vectors $\mathbf{t}_i$ representing look-up-tables, which are combined with the product. Thus, instead of just outputting a list of encryptions of X^{u_i}, Algorithm 2 outputs encryptions of $t_i(X) \cdot X^{u_i}$.

Lemma 4.1 (Correctness, noise growth and complexity of BIN-SAB). *Let the input of Algorithm 2 be* $(a(Y), b(Y)) \in \mathcal{R}_Y^2$, $\mathsf{ksk} \in \mathcal{R}_q \mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to$

$z, E_{\mathsf{ksk}})$, $\mathbf{C}_{i,j} \in \mathcal{R}_q\mathsf{GSW}^{\ell}_z(d_{i,j}, E_G)$, *and* $\mathbf{t}_i \in \mathcal{R}_q\mathsf{LWE}_z(t_i(X), E_T)$, *where* $\mathbf{t}_i$ *are possibly trivial RLWE samples and* $b(Y) = a(Y)\cdot s(Y)+e(Y)+m(Y) \bmod \langle Y^n+1, 2N\rangle$ *for some secret key* s *with Hamming weight* h *and coefficients in* $\{0,1\}$.

Then, the output satisfies $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_s(t_i(X)\cdot X^{m_i+e_i}, E_{out})$ *where*

$$E_{out} \leq \sqrt{(h+1)(\log B)\left(\ell N\beta^2 E_G^2 + N\ell_{\mathsf{ksk}}\beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_T^2}$$

Moreover, the time complexity is $O(h\cdot n\cdot \log B)$ *external products.*

Proof. Correctness follows trivially from Equation (3), since each call to MPMUL multiplies the accumulator by $Y^{\mathbf{d}[i]}$ and the inner loop subtracts $a(Y)$ in the exponent.

Also, since MPMUL executes $O(n\log B)$ external products (and some automorphisms, which have essentially the same time complexity as an external product), it is easy to see that claimed time complexity holds.

Now, from the noise growth of the homomorphic operations shown in Section 2.1, we see that the multiplications by plaintext do not increase the noise, since $\|X^{-a_i}\|_2 = \left\|X^{-b_i}\right\|_2 = 1$. Thus, only the calls to MPMUL change the noise. But from Lemma 3.1, considering that the i-th call to MPMUL has RLWE samples with E_i-subgaussian noise and GSW ciphertexts with E_G-subgaussian noise, we can see that the output has E_{i+1}-subgaussian noise where

$$E_{i+1} \leq \sqrt{(\log B)\left(\ell N\beta^2 E_G^2 + N\ell_{\mathsf{ksk}}\beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_i^2}$$

therefore, since $E_0 = E_T$, the last call to MPMUL, when $i = h$, gives us

$$E_{i+1} \leq \sqrt{(h+1)(\log B)\left(\ell N\beta^2 E_G^2 + N\ell_{\mathsf{ksk}}\beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_T^2}$$

□

4.2 Bootstrapping with ternary secrets

Now let's move to a widely used case, when the RLWE has fixed Hamming weight, h, and coefficients sampled from $\{-1, 0, 1\}$. Then, defining $\tilde{j} := \mathsf{idx}(s)$, $\mathbf{d} := \mathsf{diff}(s)$, and $r := b - a\cdot s = b - \sum_{j\in\mathsf{idx}(s)}(as_j)\odot Y^j$ as in Section 4.1, we can write

$$r = \left(\left(\left(b\odot Y^{\mathbf{d}[0]} - as_{\tilde{j}_0}\right)\odot Y^{\mathbf{d}[1]} - as_{\tilde{j}_1}\right)\cdots - as_{\tilde{j}_{h-1}}\right)\odot Y^{\mathbf{d}[h]} \tag{4}$$

The only difference when compared to Equation 3 is that we do not simply subtract a all the time. When the coefficient $s_{\tilde{j}_i}$ is -1, we have to add a instead of subtracting it. To achieve this, we include as input another encrypted control bit which is 1 if $s_{\tilde{j}_i} = 1$ and 0 otherwise. Then, by using this control bit, we use a CMux to select between $\mathsf{FHE.MultPtxt}(\mathbf{c}_i, X^{-a_i})$ and $\mathsf{FHE.MultPtxt}(\mathbf{c}_i, X^{a_i})$. We show this in detail in Algorithm 3.

Algorithm 3: Ternary Sparse Amortized Bootstrapping (TERN-SAB)

Input : an RLWE ciphertext $(a, b) \in \mathcal{R}_Y^2$ encrypting $m(Y) = \sum_{i=0}^{n} m_i Y^i$ under a key s with noise e. For $0 \leq i < n$, $a_i, b_i \in \mathbb{Z}_{2N}$

Input : GSW ciphertexts encrypting the binary decomposition of $\mathbf{d} := \mathsf{diff}(s)$, as required by MPMUL, i.e., $\mathbf{C}_{i,j} \in \mathcal{R}_q\mathsf{GSW}_z^\ell(d_{i,j})$ for $0 \leq i \leq h$, $0 \leq j < \log B$, where $\sum_{j=0}^{\log B - 1} d_{i,j} 2^j = \mathbf{d}[i]$

Input : a key $\mathsf{ksk} \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to z, E_{\mathsf{ksk}})$ for the automorphism

Input : test vectors $\mathbf{t}_0, ..., \mathbf{t}_{n-1}$ encoding LUTs as polynomials $t_i(X)$

Input : encrypted control bits $\mathbf{V}_i \in \mathcal{R}_q\mathsf{GSW}_z^\ell(b_i)$ for $1 \leq i \leq h$ where $b_i = 1$ if $s_{\tilde{j}_i} = 1$ and $b_i = 0$ otherwise

Output: a list $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_z(t_i(X) \cdot X^{m_i + e_i})$ for $0 \leq i < n$.

1 Let $\mathbf{c} := (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$ where $\mathbf{c}_i = \mathsf{FHE.MultPtxt}(\mathbf{t}_i, X^{b_i})$
2 **for** $0 \leq i \leq h - 1$ **do**
  /* Multiply by $X^{\mathbf{d}[i]}$ */
3 Let $\mathbf{C} = (\mathbf{C}_{i,1}, ..., \mathbf{C}_{i,\log n - 1})$
4 $\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
  /* Subtract or add $a(Y)$ */
5 **for** $0 \leq k \leq n - 1$ **do**
6 $\mathbf{p} \leftarrow \mathsf{FHE.MultPtxt}(\mathbf{c}_k, X^{-a_k})$
7 $\mathbf{z} \leftarrow \mathsf{FHE.MultPtxt}(\mathbf{c}_k, X^{a_k})$
8 $\mathbf{c}_k \leftarrow (\mathbf{p} - \mathbf{z}) \boxdot \mathbf{V}_k + \mathbf{z}$
9 $\mathbf{c} \leftarrow (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$
10 Let $\mathbf{C} = (\mathbf{C}_{h,1}, ..., \mathbf{C}_{h,\log n - 1})$
11 $\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
12 **return** $\mathbf{c}$

Lemma 4.2 (Correctness, noise growth and complexity of TERN-SAB). *Consider an execution of Algorithm 3 with this input:* $(a(Y), b(Y)) \in \mathcal{R}_Y^2$, $\mathbf{C}_{i,j} \in \mathcal{R}_q\mathsf{GSW}_z^\ell(d_{i,j}, E_G)$, $\mathbf{t}_i \in \mathcal{R}_q\mathsf{LWE}_z(t_i(X), E_T)$, $\mathsf{ksk} \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^{-1}) \to z, E_{\mathsf{ksk}})$, *and* $\mathbf{V}_i \in \mathcal{R}_q\mathsf{GSW}_z^\ell(b_i, E_G)$, *where* $\mathbf{t}_i$ *are possibly trivial RLWE samples and* $b(Y) = a(Y) \cdot s(Y) + e(Y) + m(Y) \bmod \langle Y^n + 1, 2N \rangle$ *for some secret key* s *with Hamming weight* h *and coefficients in* $\{0, 1\}$, $b_i \in \{0, 1\}$, *and* $b_i = 1 \Leftrightarrow \tilde{j}_i = 1$.

Then, the output ciphertexts $\mathbf{c}_i$ *satisfy* $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_s(t_i(X) \cdot X^{m_i + e_i}, E_{out})$ *where*

$$E_h \leq \sqrt{h\ell N\beta^2 E_G^2 + (h+1)(\log B)\left(\ell N\beta^2 E_G^2 + N\ell_{\mathsf{ksk}}\beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_T^2}$$

Moreover, the time complexity is $O(h \cdot n \cdot \log B)$ *external products.*

Proof. The proof is almost the same as Lemma 4.1.

Correctness follows from Equation 4 and from the fact that line 8 subtracts $s_{\tilde{j}_k} \cdot a_k$, as expected, since $s_{\tilde{j}_k} = 1$ implies $\mathbf{V}_k$ encrypting 1, which means that $\mathbf{c}_k \cdot X^{-a_k}$ is selected, otherwise, $\mathbf{V}_k$ encrypts 0 and $\mathbf{c}_k$ is updated to $\mathbf{z} = \mathbf{c}_k \cdot X^{a_k}$.

Also, it only performs hn additional external products (line 8) compared to Algorithm 2, thus, the time complexity is $O(hn \log B + hn) = O(hn \log B)$ external products.

As for the noise, consider that at the i-th iteration, the vector $\mathbf{c}$ has ciphertexts with E_i-subgaussian noise. After executing MPMUL we have $\hat{E} \leq \sqrt{(\log B)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_i^2}$. The calls to $\mathsf{FHE.MultPtxt}$ do not increase the noise since $\|X^{a_k}\|_2 = \|X^{-a_k}\|_2 = \left\|X^{-b_i}\right\|_2 = 1$. Then, the external product in line 8 yields

$$\begin{aligned} E_{i+1} &\leq \sqrt{\ell N \beta^2 E_G^2 + b_k \cdot \hat{E}^2} \\ &\leq \sqrt{\ell N \beta^2 E_G^2 + (\log B)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_i^2} \end{aligned}$$

Thus, after h iterations, we have

$$E_h \leq \sqrt{h \ell N \beta^2 E_G^2 + h(\log B)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_i^2}$$

and the final call to MPMUL gives us

$$E_{out} \leq \sqrt{(\log B)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_h^2}$$

which is the claimed bound since $E_0 = E_T$. □

4.3 Bootstrapping with general sparse secrets

Finally, let's consider a more general case where the RLWE secret has h nonzero coefficients, all sampled from $[\![-\rho, \rho]\!]$, where ρ is any integer larger than or equal to 1. As in Section 4.1, define $\tilde{j} := \mathsf{idx}(s)$ and $\mathbf{d} := \mathsf{diff}(s)$. Then, we can write $b - a \cdot s$ exactly as in Equation 4, but with $s_{\tilde{j}_1} \in [\![-\rho, \rho]\!]$.

To evaluate this equation efficiently, we could use MPMUL to perform the multiplications by $Y^{\mathbf{d}[i]}$, then homomorphically subtract $a s_{\tilde{j}_i}$. For the subtraction, we could have GSW ciphertexts encrypting $X^{s_{\tilde{j}_i}}$, use GSW automorphism [GPvL23] to obtain GSW encryptions of $X^{-a_k s_{\tilde{j}_i}}$ for each coefficient a_k of a, then use external products to accumulate $X^{-a_k s_{\tilde{j}_i}}$, effectivelly subtracting $a s_{\tilde{j}_i}$ in the exponent. However, one GSW automorphism costs essentially the same as $O(\log \lambda)$ regular RLWE automorphisms.

Thus, we evaluate this formula differently, using two RLWE automorphisms, as proposed in [BDF18]. Firstly, we apply the automorphisms $X \mapsto X^{-a_k^{-1}}$, for each coefficient a_k, to the RLWE ciphertexts output by MPMUL. Then, we multiply by $Y^{\tilde{j}_i}$ using external product. Finally, we apply the automorphisms $X \mapsto X^{-a_k}$. This can be viewed as starting with some polynomial $u(Y)$ (with coefficients encrypted in the exponent of X), then obtaining a new polynomial $v(Y)$ where each coefficient v_i equals $-u_i \cdot a_i^{-1} \bmod 2N$, then computing $v(Y) + s_{\tilde{j}_i} \cdot Y^{\tilde{j}_i}$ (with n external products), and finally using the second automorphism to obtain $w(Y)$ with coefficients $w_i = -v_i \cdot a_i - a_i \cdot s_{\tilde{j}_i} = u_i - a_i \cdot s_{\tilde{j}_i}$.

Hence, with one external product and two RLWE automorphisms per coefficient instead of one GSW automorphism per coefficient, we are able to homomorphically subtract $a(Y) \cdot s_{\tilde{j}_i}$, as required in Equation 4. Then, multiplying by $Y^{\mathbf{d}[i]}$ is performed as before, using MPMUL. This gives us Algorithm 4.

Algorithm 4: General Sparse Amortized Bootstrapping (ρ-SAB)

Input : an RLWE ciphertext $(a, b) \in \mathcal{R}_Y^2$ encrypting $m(Y) = \sum_{i=0}^{n} m_i Y^i$ under a key s with noise e. For $0 \le i < n$, $a_i, b_i \in \mathbb{Z}_{2N}$ and a_i is odd

Input : GSW ciphertexts encrypting the binary decomposition of $\mathbf{d} := \mathsf{diff}(s)$, as required by MPMUL, i.e., $\mathbf{C}_{i,j} \in \mathcal{R}_q\mathsf{GSW}_z^\ell(d_{i,j})$ for $0 \le i \le h$, $0 \le j < \log B$, where $\sum_{j=0}^{\log B - 1} d_{i,j} 2^j = \mathbf{d}[i]$

Input : key-switching keys $\mathsf{ksk}_i \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^i) \to z, E_{\mathsf{ksk}})$ for the automorphism $X \mapsto X^i$ for $i \in \{1, 3, 5, ..., 2N-1\}$

Input : test vectors $\mathbf{t}_0, ..., \mathbf{t}_{n-1}$ encoding LUTs as polynomials $t_i(X)$

Input : coefficients of s encrypted in the exponent, i.e., $\mathbf{V}_i \in \mathcal{R}_q\mathsf{GSW}_z^\ell(X^{s_{\tilde{j}_i}})$ for $0 \le i \le h-1$

Output: a list $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_z(t_i(X) \cdot X^{m_i + e_i})$ for $0 \le i < n$.

Let $\mathbf{c} := (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$ where $\mathbf{c}_i = \mathsf{FHE.MultPtxt}(\mathbf{t}_i, X^{b_i})$
for $0 \le i \le h-1$ **do**
  /* Multiply by $X^{\mathbf{d}[i]}$ */
  Let $\mathbf{C} = (\mathbf{C}_{i,1}, ..., \mathbf{C}_{i,\log n - 1})$
  $\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
  /* Subtract $a(Y) \cdot s_{\tilde{j}_i}$ */
  for $0 \le k \le n-1$ **do**
    $\mathbf{y} \leftarrow \mathsf{FHE.Automorphism}(\mathbf{c}_k, -a_k^{-1})$
    $\mathbf{w} \leftarrow \mathbf{y} \boxdot \mathbf{V}_i$
    $\mathbf{c}_k \leftarrow \mathsf{FHE.Automorphism}(\mathbf{w}, -a_k)$
  $\mathbf{c} \leftarrow (\mathbf{c}_0, ..., \mathbf{c}_{n-1})$
Let $\mathbf{C} = (\mathbf{C}_{h,1}, ..., \mathbf{C}_{h,\log n - 1})$
$\mathbf{c} \leftarrow \text{MPMUL}(\mathbf{c}, \mathbf{C}, \mathsf{ksk})$
return $\mathbf{c}$

As a technicality, the automorphisms $X \mapsto X^k$ are only defined for odd k. Therefore, in Algorithm 4, we assume that all input coefficients a_i are odd. This can be achieved with little noise growth using modulus switching [LMK+23].

Lemma 4.3 (Correctness, noise growth and complexity of ρ-SAB). *Consider Algorithm 4 with the following input:* $(a(Y), b(Y)) \in \mathcal{R}_Y^2$, $\mathsf{ksk}_i \in \mathcal{R}_q\mathsf{KS}^{\beta_{\mathsf{ksk}}}(z(X^i) \to z, E_{\mathsf{ksk}})$, $\mathbf{C}_{i,j} \in \mathcal{R}_q\mathsf{GSW}_z^\ell(d_{i,j}, E_G)$, $\mathbf{V}_i \in \mathcal{R}_q\mathsf{GSW}_z^\ell(X^{s_{\tilde{j}_i}}, E_G)$, *and* $\mathbf{t}_i \in \mathcal{R}_q\mathsf{LWE}_z(t_i(X), E_T)$, *where* $\mathbf{t}_i$ *are possibly trivial RLWE samples,* $b(Y) = a(Y) \cdot s(Y) + e(Y) + m(Y) \bmod \langle Y^n + 1, 2N \rangle$ *for some secret key* s *with Hamming weight* h *and coefficients in* $[\![-\rho, \rho]\!]$, *and the coefficients of* $a(Y)$ *are odd numbers.*

Then, $\mathbf{c}_i \in \mathcal{R}_q\mathsf{LWE}_s(t_i(X) \cdot X^{m_i + e_i}, E_{out})$ *for the output* $(\mathbf{c}_0, ..., \mathbf{c}_{n-1})$, *where*

$$E_{out} \le \sqrt{(h+1)(\log B + 2)\left(\ell N \beta^2 E_G^2 + N \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_T^2}$$

Moreover, the time complexity is $O(h \cdot n \cdot \log B)$ *external products.*

Proof. Essentially the same as Lemma 4.1 and Lemma 4.3. □

5 Packing Key Switching

Exploiting a somewhat similar idea to our sparse multiplication algorithm, we introduce a new approach to perform packing key switching that introduces lower noise at the cost of more multiplications. Let $c_i = (a_i, b_i) \in \mathbb{Z}^N \times \mathbb{Z}$ for $i \in [0, n)$ be a list of n LWE ciphertexts, the first step of a packing key switching consists of arranging the coefficients of LWE ciphertexts into a matrix:

$$\left(\begin{pmatrix} a_{0,0}, & a_{0,1}, & \dots, & a_{0,N-1} \\ \dots & \dots & \dots & \dots \\ a_{n-1,0}, & a_{n-1,1}, & \dots, & a_{n-1,N-1} \end{pmatrix}, \begin{pmatrix} b_0 \\ \dots \\ b_{n-1} \end{pmatrix} \right) \in \mathbb{Z}_q^{n \times N} \times \mathbb{Z}_q^{n \times 1}$$

Let $N' \geq n$ be the dimension of the output RLWE and let $f : \mathbb{Z}_q^n \mapsto \mathbb{Z}_q[X]/(X^{N'} + 1)$, given by $f(x_0, ..., x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i X^i$ be a function defining a coefficient embedding, one can produce a module-LWE ciphertext of dimension N' and rank N by applying f to every column of the matrices:

$$(a', b') = \left(\sum_{i=0}^{n-1} a_{i,0} X^i, \sum_{i=0}^{n-1} a_{i,1} X^i, \dots, \sum_{i=0}^{n-1} a_{i,N-1} X^i\right), \left(\sum_{i=0}^{n-1} b_i X^i\right) \in \mathcal{R}_q^N \times \mathcal{R}_q$$

Usual RLWE key switching algorithms can now be used to switch this ciphertext to different parameters and keys. Traditional techniques [CGGI16, MS18, GPvL23] compute $b' - \langle a', s \rangle$ for some secret key $s \in \mathbb{Z}_q^N$ by performing $O(N)$ multiplications between the polynomials in a' and key switching keys encrypting s. This process introduces noise $O(\sqrt{Nn})$ and requires $O(NN')$ evaluation key size. We introduce an algorithm that requires $O(hN)$ multiplications, but only introduces noise $O(\sqrt{h \log(N)} \sqrt{N'})$ and requires $O(h \log(N) N')$ evaluation key size. Algorithm 5 presents our solution.

The basic idea behind it is to use a CMUX tree [CGGI17] to select only the elements of a' corresponding to the nonzero positions of s. Naively, we could run a CMUX tree h times (one per nonzero position) over the entire a', which leads to the asymptotic costs mentioned above. However, this process can be accelerated by assuming some structure in the key (which happens with very high probability), similarly as we do with our amortized bootstrapping algorithm. Let $s \in \mathbb{Z}_q^N$ be a vector of Hamming Weight h such that $\mathsf{idx}(s)$ is uniformly distributed in $\mathbb{Z}_N^h$. We choose parameters $B \leq N$ and $k \in [1, h]$, such that $\Pr\left[\mathsf{HW}(s_{Bi,\dots,B(i+1)-1}) > k\right]$ is small for all $i \in [0, N/B)$. Based on this, we divide a' into N/B blocks of size B, and we use a CMUX tree to select k positions from each block (Lines 8 to 11 of Algorithm 5). The selected positions of a' are then multiplied by their respective values in the key (Line 12 of Algorithm 5). The goal is to select all positions corresponding to nonzero elements of key, which we call *"relevant positions"*. If a block has less than k relevant positions, we select any other element, as it will be multiplied by an encryption of 0 in Line 12. Conversely, if a block has more than k relevant positions, the algorithm would fail, which we avoid by choosing (B, k) that would lead to negligible probability of failure. We detail the key generation process in Algorithm 7 in

Algorithm 5: Packing key switching for sparse secrets

Input : a list of $n \leq N'$ LWE ciphertexts $\mathbf{c}_i = (\mathbf{a}_i, b_i) \in \mathsf{LWE}^q_{\mathbf{s}}(m_i) \subset \mathbb{Z}_q^N \times \mathbb{Z}_q$, for $i \in [0, n)$.
Input : parameters $k \in [1, h]$ and $B \leq N$ s.t. $B|N$
Input : two vectors (V', C') containing, respectively, GSW encryptions of V and of the binary decomposition of C, where (V, C) are produced by Algorithm 7
. **Output:** a ciphertext $c' \in \mathcal{R}_q\mathsf{LWE}_{s'}(\sum_{i=0}^{n-1} m_i)$
`/* Pack LWE ciphertexts into an MLWE of rank` N `*/`
$f : \mathbb{Z}_q^n \mapsto \mathcal{R}_q$, given by $f(x_0, ..., x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i X^i$
$c' \leftarrow (0, f([b_i]_{i=0,...,n-1}))$
for $i \leftarrow 0$ **to** N **do**
  $\hat{a}_i \leftarrow f([a_{i,j}]_{j=0,...,n-1})$
`/* Key switching */`
for $j \leftarrow 0$ **to** N/B **do**
  for $i \leftarrow 0$ **to** k **do**
    `/* CMUX tree: select a coeff. in block` j `*/`
    $\tilde{a} \leftarrow \hat{a}$
    for $t \leftarrow 0$ **to** $\log B$ **do**
      $z \leftarrow 2^{\log B - t - 1}$
      for $r \leftarrow 0$ **to** z **do**
        $\tilde{a}_{jB+r} \leftarrow (\tilde{a}_{jB+r+z} - \tilde{a}_{jB+r}) \boxdot C_{jk+i,\log(B)-t-1} + \tilde{a}_{jB+r}$
    `/* Subtract selected position times V */`
    $c' \leftarrow c' - \tilde{a}_{jB} \boxdot V_{jk+i}$
Return c'

Table 2. Comparison between packing methods. Cost measured in the number of RLWE multiplications; Key size is given in the number of $\mathbb{Z}_q$ elements. The parameter ℓ is the gadget decomposition degree. We assume $n = N' = N$.

Method	Cost	Noise overhead	Key size
Traditional	ℓN	$N\sqrt{\ell}$	$2\ell N^2$
[CDKS21]	$\ell N \log(N)$	$N\sqrt{\ell \log(N)}$	$2\ell N \log(N)$
This work	$2hN\ell$	$\sqrt{2\ell hN \log(N)}$	$4\ell hN \log(N)$
	$2kN\ell$	$N\sqrt{2\ell(k/B)\log(B)}$	$4\ell N^2(k/B)\log(B)$

Appendix A. To further improve performance, we can also select (B, k) such that $\Pr\left[\mathsf{HW}(s_{Bi,...,B(i+1)-1}) > k\right]$ is not negligible and perform rejection sampling (the impacts on security will be addressed in Section 6.2). The final cost is $O(Nk)$ GSW multiplications with noise $O(\sqrt{k(N/B)\log B}\sqrt{N'})$ and evaluation key size $O(k(N/B)\log(B)N')$. Table 2 compares with alternative approaches. Generally, we see our new method as an interesting theoretical result, but of limited practical applicability due to its increased cost. Specifically, it enables noise overhead $\tilde{O}(\sqrt{hN})$ instead of the typical $O(N)$ of existing methods, which may become relevant as the understanding of LWE sparsity advances.

6 Our amortized functional bootstrapping

Once we have the established our main tools, namely, the LWE-to-RLWE packing and the homomorphic evaluations of the first step of the RLWE decryption, we can proceed to presenting our bootstrapping algorithm. As illustrated in Figure 2, we have a function bootstrapping, thus, n arbitrary functions can be chosen. If they are private, one generate the corresponding test vectors as actual RLWE encryptions, otherwise, the test vectors are trivial and noiseless RLWE samples with the polynomial corresponding to the function presented in clear.

The bootstrapping starts with n LWE ciphertexts, which are packed, and modulus-switched, and key-switched into a single RLWE ciphertext (a, b) where $b(Y) = a(Y) \cdot s(Y) + e(Y) + m(Y) \mod \langle Y^n + 1, 2N \rangle$, where s is a secret key with Hamming weight h. Then, depending if the coefficients of s are binary, ternary, or between $-\rho$ and ρ, we use either BIN-SAB, TERN-SAB, or ρ-SAB to homomorphically compute $u(Y) := b(Y) - a \cdot s(Y) = e(Y) + m(Y) \mod \langle Y^n + 1, 2N \rangle$. The result is a set of RLWE ciphertexts encrypting

$$t_i(X)X^{u_i} = t_i(X)X^{e_i+m_i} = f_i(m_i) + X \cdot g_i(X)$$

where $g_i(X)$ is some polynomial with degree less than $N-1$. Finally, the constant term, i.e., $f_i(m_i)$ is extracted into an LWE ciphertext. We show this thoroughly in Algorithm 6.

As a practical optimization, since the noise overhead and the running time of the bootstrapping are linear in the Hamming weight of the secret key, before the main part of the bootstrapping, we run a key switching to obtain an RLWE ciphertext under a low Hamming-weight key. This is denoted by HW-ReducingKeySwitching in our algorithm, and we discuss it in more detail in Section 7.1.

Algorithm 6: Amortized functional bootstrapping

Input : $\mathbf{c}_i \in \mathsf{LWE}^q_{\mathbf{s}}(m_i)$ s.t. $m_i \in \mathbb{Z}_{p/2}$ for $1 \leq i \leq N$
Input : a set of N functions $f_i : \mathbb{Z}_{p/2} \mapsto \mathbb{Z}_{p/2}$ for $1 \leq i \leq N$
Input : bootstrapping key BSK required by BIN-SAB, TERN-SAB, or ρ-SAB
Input : packing and key-switching keys PCK and KSK
Output: $\mathbf{c}_i^{(out)} \in \mathsf{LWE}^q_{\mathbf{s}}(f_i(m_i))$ for $1 \leq i \leq N$

For each f_i, define a test vector $\mathbf{t}_i \in \mathcal{R}^2$
$\mathbf{t} \leftarrow (\mathbf{t}_1, ..., \mathbf{t}_N)$
$\mathbf{c}_{\mathsf{pck}} \leftarrow \mathsf{PackingKeySwitching}(\mathbf{c}_1, ..., \mathbf{c}_n, \mathrm{PCK})$
$\mathbf{c} \leftarrow \mathsf{HW\text{-}ReducingKeySwitching}(\mathbf{c}_{\mathsf{pck}}, \mathrm{KSK})$
$\hat{\mathbf{c}} \leftarrow \mathsf{SAB}(\mathbf{c}, \mathbf{t}, \mathrm{BSK})$ `// Call BIN-, TERN-, or ρ-SAB`
for $1 \leq i \leq N$ **do**
  $\mathbf{c}_i^{(out)} \leftarrow \mathsf{LWE.Extract}(\hat{\mathbf{c}}_i)$
Return $\mathbf{c}_1^{(out)}, ..., \mathbf{c}_N^{(out)}$

Since extraction does not increase the noise, at the end, we obtain n LWE ciphertexts encrypting $f_i(m_i)$ with the same noise output by the chosen SAB algorithm. This gives us Lemma 6.1

Lemma 6.1 (Noise overhead of amortized functional bootstrapping). *Consider the same notation used in Lemma 4.2. Algorithm 6 runs in time $O(nh \log B)$ external products and outputs LWE ciphertexts with E-subgaussian noise where*

$$E \in O\left(\sqrt{Nh \log B \left(\ell \beta^2 E_G^2 + \ell_{\mathsf{ksk}} \beta_{\mathsf{ksk}}^2 E_{\mathsf{ksk}}^2\right) + E_T^2}\right)$$

Proof. The proof follows trivially from the time complexity and noise overhead of BIN-SAB, TERN-SAB, and ρ-SAB, and also from the fact that they dominate the running time of the bootstrapping and their output is basically the output of the bootstrapping (except by the extraction procedure, but it does not change the noise). □

Corollary 6.2. *Assuming $\beta, \beta_{\mathsf{ksk}}, E_G, E_{\mathsf{ksk}} \in O(1)$, Algorithm 6 has amortized time complexity of $O(h \log B)$ external products per message and noise overhead $\tilde{O}\left(\sqrt{Nh \log B}\right)$.*

Proof. Since $\beta, \beta_{\mathsf{ksk}} \in O(1)$, we have $\ell, \ell_{\mathsf{ksk}} \in (\log q) = O(\log \lambda)$. Then, the parameter E from Lemma 6.1 is $\tilde{O}(\sqrt{Nh \log B})$. Also, as we refresh n messages per bootstrapping, from Lemma 6.1, the amortized complexity is $O((nh \log B)/n) = O(h \log B)$ external products. □

6.1 The choice of the parameter B and efficiency

Recall that the main step of our bootstrapping process an RLWE ciphertext $(a, b) \in \mathbb{Z}_{2N}[Y]/\langle Y^n + 1\rangle$ using some key $s(Y)$ with Hamming weight h. We define $\tilde{j} = \mathsf{idx}(s)$ as the sequence of exponents of the nonzero coefficients of s. For instance, if $s(Y) = 3 \cdot Y - 1 \cdot Y^3 + 3 \cdot Y^8$, then $\tilde{j} = (1, 3, 8)$. Also, define $\mathbf{d} := \mathsf{diff}(s) \in [\![-n, n]\!]^h$ as $\mathbf{d}[0] = -\tilde{j}_0$, $\mathbf{d}[h] = -\tilde{j}_{h-1}$, and $\mathbf{d}[i] = \tilde{j}_{i-1} - \tilde{j}_i$ for $1 \leq i \leq h-1$. Then, the algorithms presented in Section 4 require us to encrypt the bits corresponding to the binary decomposition of each $\mathbf{d}[i]$. Thus, we introduced a parameter B representing an upper bound to $|\mathbf{d}[i]|$ for $1 \leq i \leq h-1$, and both the time complexity and the noise overhead of our bootstrappping depend on $\log B$.

A trivial and over pessimist choice for B is n, since each $\tilde{j}_i \in [\![0, n-1]\!]$ and $\tilde{j}_{i+1} > \tilde{j}_i$, implying that $|\mathbf{d}[i]| = |\tilde{j}_i - \tilde{j}_{i+1}| \leq n - 1$.

However, we expect the h nonzero coefficients of s to be well spread over the n possible positions, in other words, we expect $|\mathbf{d}[i]| \in O(n/h)$ for $1 \leq i \leq h-1$. Hence, by choosing $B \in O(n/h)$, we obtain from Corollary 6.2 that our bootstrapping requires $O(h \log(n/h))$ external products per refreshed message and has $\tilde{O}(\sqrt{Nh \log(n/h)})$ noise overhead.

Moreover, a typical choice of parameters for FHE is $N, n, h \in O(\lambda)$, which gives us $O(\log(n/\lambda)) = O(1)$, implying that our amortized time complexity, in

terms of external products, is only $O(h) = O(\lambda)$ while our noise overhead is $\tilde{O}(\lambda)$. But if one were willing to choose $n \in O(\lambda \log^k \lambda)$ and $h \in O(\lambda / \log \lambda)$, the required number of external products per refreshed message would be

$$O(h \log(n/h)) = O\left(\frac{\lambda(k+1)\log(\log \lambda)}{\log \lambda}\right) = o(\lambda)$$

that is, sublinear in λ, while the noise overhead would still be $\tilde{O}(\sqrt{Nh\log(n/h)}) = \tilde{O}(\lambda)$, i.e., quasilinear in λ.

6.2 The choice of the parameter B and security

In practice, we fix a concrete value for $B \approx N/h$ and, during the key generation, we apply rejection sampling if the sampled key s does not satisfy $|\mathbf{d}[i]| < B$ for $1 \leq i \leq h-1$, where $\mathbf{d} := \mathsf{diff}(s)$. We argue that if the probability of a random key s satisfy $|\mathbf{d}[i]| < B$ is upper bounded by $1/2^\delta \in \mathbb{R}$, then, any attacker $\mathcal{A}$ against the original scheme, without rejection sampling, would already have probability $1/2^\delta$ of interacting with a key that is accepted by our rejection sampling. Therefore, if $\mathcal{A}$ could break the CPA-security of our scheme with rejection sampling with non-negligible probability $a(\lambda)$, $\mathcal{A}$ would essentially have probability $a(\lambda)/2^\delta$ of breaking the CPA-security of the same scheme without rejection sampling. But $a(\lambda)/2^\delta$ would still be non-negligible if $\delta \in \omega(\lambda)$.

By contrapositive, if no attacker can break the CPA-security of the original scheme with probability less than some $a(\lambda)$, then no attacker can break the CPA-security of the modified scheme (with rejection sampling) with probability less than $a(\lambda) \cdot 2^\delta$.

Hence, in order to adjust the concrete parameters we pick, we estimate the probability $1/2^\delta$ and subtract δ from the security level the original scheme achieves. In discuss this in detail in Section 7.1.

7 Practical results

7.1 Parameter selection

Our bootstrapping requires 3 sets of evaluation keys, two for key switchings and one for the bootstrapping itself. Table 3 shows the parameters we select for them. In addition to the parameters described in the previous sections, each key has also its associated values ℓ_{A} and β corresponding to the degree and base of the approximate gadget decomposition [CGGI16] they are used with. These decompositions are standard in TFHE-related literature [CGGI16, CGGI17], but, for completeness, we present the full equations for their noise analysis in Appendix B. Parameters are selected as follows.

– **Repacking key:** There are no specific requirements to its distribution, but we choose sparse ternary for convenience. We benchmark it for $N \in \{2^{11}, 2^{12}, 2^{13}\}$, as this corresponds to the number of messages to be bootstrapped at once.

Table 3. Parameters for repacking and bootstrapping. All keys are sparse ternary and $q = 2^{64}$.

Key	Key use	N	h	ℓ_A	$\log_2(\beta)$	q/σ	λ
RPC_1	Repacking	2048	256	2	14	2^{44}	143.1
RPC_2		4096	256	2	14	2^{44}	287.2
RPC_3		8192	256	2	14	2^{44}	689.3
FBS_1	Bootstrapping	2048	512	1	23	2^{50}	128.9
FBS_2		8192	512	1	22	2^{51}	511.8

We do not optimize parameters such as Hamming Weight or security level, as the repacking takes only 1 to 5% of the execution time of our approach, and, hence, there's little to no incentive to optimize it.

– **Bootstrapping key:** If generic repacking methods are used, there is also no requirements to its distribution, and, again, we choose sparse ternary for convenience. For our new packing KS method (Section 5), a higher level of sparsity in the bootstrapping key would improve performance and noise. In practice, however, we estimate our new method would only reduce the noise overhead of the repacking by a factor of at most 4, which would have no impact in the overall protocol. Therefore, we choose to use generic keys for bootstrapping and let the choice of repacking method arbitrary. We define keys for dimensions $N \in \{2^{11}, 2^{13}\}$ to allow the bootstrapping of small (up to 6 bits) and large (8 bits) messages, respectively. The keys for automorphisms are generated using the same parameters as the bootstrapping keys.
– **Hamming-weight-reducing key switching key:** Our bootstrapping algorithms could work directly over ciphertexts encrypted under RPC and, given our parameters, they would likely already be faster than non-amortized versions of the bootstrapping. This, however, would be far from optimal. Once the LWE ciphertexts are packed in a single RLWE, we can significantly reduce the Hamming Weight by applying another key switching. This process follows a similar idea to the *sparse-secret encapsulation* technique from [BTH22], which key-switches the ciphertext to a key of much smaller Hamming Weight (but also much smaller modulus to keep the security level). This leads us to the Parameters presented in Table 4. Since we need to minimize noise as much as possible, all HW-reducing key switching use binary decomposition with degree ℓ_A indicated by Table 4. Our method requires sparsity, but it has no requirements for the distribution of the non-zero positions. Therefore, we define parameters for sparse binary, ternary, and balanced arbitrary in $\mathbb{Z}_8$ keys.

Security Level For all keys in Tables 3 and 4, the security level is defined by the smallest result of the Lattice Estimator [APS15] using the default reduction model. We only use parameters for which the Estimator: 1) is able to estimate security under all available attacks without errors, and 2) gives results that are

Table 4. Parameters for Binary, Ternary, and Arbitrary in $\mathbb{Z}_8$ keys. For all keys, $\log_2(\beta) = 1$ and $q = 2^{64}$. Parameters B and δ are chosen as detailed in Section 6.2, and the security level (λ) is already adjusted accordingly.

Key	Type	N	h	ℓ_A	q/σ	B	δ	λ
B2	Binary	2048	18	12	2^{15}	2^9	0.2	128.3
B4		2048	21	14	2^{17}	2^8	2.6	128.7
B6		2048	28	16	2^{20}	2^8	1.2	129.1
B8		2048	33	19	2^{23}	2^8	0.7	129.1
T2	Ternary	2048	17	12	2^{15}	2^9	0.2	128.8
T4		2048	21	14	2^{17}	2^8	2.6	128.3
T6		2048	27	16	2^{20}	2^8	1.3	128.1
T8		2048	30	19	2^{23}	2^8	0.9	127.9
A2	Arbitrary $\mathbb{Z}_8$	2048	17	12	2^{15}	2^9	0.2	> 128.8
A4		2048	24	14	2^{18}	2^8	1.8	> 134.3
A6		2048	30	16	2^{23}	2^8	0.9	> 127.9

consistent with adjacent parameters (*e.g.*, parameters for which slightly increasing or decreasing the values of q, N, or σ results in the expected impact on security). This precaution is taken as the estimator may fail or present inconsistencies for some sets of small parameters. We note that some previous literature on bootstrapping suggest extrapolating results for these cases [BTH22], which could be leveraged to further improve performance, but we choose to remain conservative for this work. The results of Table 4 are also already adjusted based on the loss (δ) imposed by the rejection sampling (Section 6.2). See Appendix C for details on how δ is computed. For sparse Arbitrary keys, we estimate security supposing they are sparse ternary, as the Estimator doesn't support them.

Probability of failure Consider an RLWE ciphertext with Gaussian-distributed noise e with standard deviation σ encrypting $m = \sum_{i=0}^{N-1} \Delta m_i X^i$, such that $m_i \in \mathbb{Z}_{2^k}$ and $\Delta = q/2^{k+1}$. Notice that we are leaving one bit of padding for the message, which is detailed in Remark 7.1. A message $m_i \in \mathbb{Z}_{2^k}$ is correctly bootstrapped with probability $\Pr[e_i < q/2^{k+2}] = \mathsf{erf}\left(\frac{q/2^{k+2}}{\sigma\sqrt{2}}\right)$ [DM15, CGGI16, BST20, GBA21, GPvL23], where erf is Gauss error function. We implemented a script to estimate the input σ for our bootstrapping based on our average case noise analysis (Appendix B), which we validated by measuring the error variance at different steps of our implementation.

Negacyclicity Our work is the first to present an amortized bootstrapping method with polynomial noise overhead that can operate strictly over rings defined by power-of-two cyclotomics, without requiring other rings. Performance-wise, this is a major advantage, as it enables the use of standard libraries featuring highly-optimized NTT or FFT-based arithmetic. On the other hand, this

Table 5. Sparse amortized bootstrapping for binary (B), ternary (T), and arbitrary (A) keys. All results are for batches of 2048 messages and use RPC_1 for repacking.

Keys	Precision (bits)	Bootstrapping Key Size[a]	Failure probability	Total Time	Amortized Time
B2+FBS_1	2	10.16MB	2^{-121}	5.1s	2.49ms
B4+FBS_1	4	10.53MB	2^{-99}	5.2s	2.52ms
B6+FBS_1	6	14.03MB	2^{-67}	6.8s	3.32ms
B8+FBS_2	8	66.13MB	2^{-63}	54.3s	26.51ms
T2+FBS_1	2	10.66MB	2^{-121}	5.3s	2.57ms
T4+FBS_1	4	11.84MB	2^{-99}	5.7s	2.78ms
T6+FBS_1	6	15.22MB	2^{-69}	7.2s	3.52ms
T8+FBS_2	8	67.63MB	2^{-68}	54.2s	26.46ms
A2+FBS_1	2	74.63MB	2^{-115}	6.15s	3.01ms
A4+FBS_1	4	77.50MB	2^{-78}	7.59s	3.70ms
A6+FBS_2	6	1.07GB	2^{-77}	72.48s	35.39ms

[a] Includes all keys required inside the bootstrapping itself, including the ones for automorphisms. See Section 7.2 for external procedures, such as the repacking key switching.

brings the usual problems related to power-of-two cyclotomics, such as negacyclicity. In short, an AP-style functional bootstrapping defined over $X^N + 1$ evaluates only negacyclic functions, *i.e.* functions $f : \mathbb{Z}_{2N} \mapsto \mathbb{Z}_{2N}$ such that $f(x + N) = -f(x)$. To overcome this limitation, one evaluates a so called *"half-domain"* functional bootstrapping, where one bit of padding is added at the beginning of the message. Methods to enable full-domain bootstrapping are broadly available in the literature [GBA24]. All the results presented in this paper already consider limitation. To give a concrete example, a level of precision of 4 bits means that one can evaluate an arbitrary function $f_{\mathsf{arbitrary}} : \mathbb{Z}_{2^4} \mapsto \mathbb{Z}_{2^4}$ or a negacyclic function $f_{\mathsf{negacyclic}} : \mathbb{Z}_{2^5} \mapsto \mathbb{Z}_{2^5}$. This is also true for the results from TFHE-rs [Zam22], but not for [GPvL23], which avoids negacyclicity by working over circulant rings.

7.2 Main results

We implemented our solution in `C` using the MOSFHET library [GBA24], compiled it with Intel(R) `DPC++` compiler v2025.0.4, and benchmarked it on an `r7i.metal-24xl` instance on AWS with an Intel Xeon Platinum 8488C at 2.4GHz (3.8GHz boost) and 768GB of memory. All results are for single-threaded execution and each value is the average of 10 executions. We measured the results for TFHE-rs [Zam22] and GPV23 [GPvL23] (Table 1) in the same environment using the benchmarking tools they provided. Table 5 shows our main results.

Compared to TFHE-rs, which is a state-of-the-art implementation for non-amortized polynomial-noise bootstrapping, we achieve gains of up to 41.5 times whiles using bootstrapping keys that are up to 50.4 time smaller. A complete comparison is presented in Table 1 (in the Introduction). Interestingly, contrary

to other GINX-style bootstrapping methods, our approach is almost not impacted by the change from binary to ternary secrets. In fact, ternary secrets generally present better probability of failure and are even slightly faster for 8-bit messages. The bootstrapping keys are also only marginally bigger (less than 10%) than binary ones. This is achieved thanks to the fact that ternary secrets enable a slightly smaller Hamming Weight while the performance of our MPMul algorithm does not depend on the distribution of the non-zero positions. For arbitrary keys in $\mathbb{Z}_8$, on the other hand, there are noticeable slowdowns, loss of precision, and increase in the size of the keys. Those are explained by three main factors. Firstly, the Galois automorphisms required by Algorithm 4 rely on large key switching keys and is two times more expensive than the CMUX required by the ternary keys. Secondly, we don't have ways of optimizing parameters for these keys (as the Lattice Estimator doesn't support their distributions), which leads us to work on a security level that is much higher than needed. Finally, their increased norm increases the noise generated by the mod switching that precedes the bootstrapping, reducing the bootstrapping precision by the same amount. Notice that since the key is balanced, we loose $\sqrt{8/2} = 2$ bits of precision for $\mathbb{Z}_8$.

Intra-coefficient batching The most common methods for batched computation in RLWE schemes are based on encoding (or *packing*) different messages into different coefficients of some encrypted polynomial or in different *"slots"* provided by some isomorphism of the plaintext ring. It is also possible, however, to pack multiple messages inside the same coefficient of a polynomial. Specifically, each k-bit coefficient of a polynomial can be used to pack up to k/n messages with n bits each. For example, a coefficient in $\mathbb{Z}_{15}$ could pack two messages $m_0 \in \mathbb{Z}_3$ and $m_1 \in \mathbb{Z}_5$ using the Chinese Remainder Theorem. Exploiting radix-based encoding, a coefficient in $\mathbb{Z}_{2^4}$ could also pack $m_0, m_1 \in \mathbb{Z}_{2^2}$ as given by $f(m_0, m_1) \mapsto m_0 + m_1 2^2$. This restricts the use of some arithmetic operations, but works generically for bootstrapped-based computation by relying on techniques such as the BML [CLOT21] multi-value bootstrapping, which allows a k-bit functional bootstrapping to evaluate n different k/n-bit functions at once. We refer to this type of packing as *"intra-coefficient batching"*. Notice that LWE-based bootstrapping techniques, such as [CGGI16], can also exploit it. Considering that, we present Table 6, which shows the optimal times for our method and TFHE-rs considering this type of batching.

Packing Key Switching Our methods enable different choices of packing algorithms, as discussed in Section 5. Performance-wise, however, this choice has little impact in the overall protocol, and, therefore, our implementation and all numbers reported in Table 5 consider the use of the traditional packing key switching. Conversely, this choice may become relevant if the goal is, for example, to minimize the size of evaluation keys. For the traditional packing KS, our repacking key takes 128MB for 2 to 6-bit messages, and 512MB for 8-bit messages. We estimate our new packing algorithm (Section 5) could reduce these

Table 6. Bootstrapping of 2-bit messages using intra-coefficient batching.

	Number of Messages	Total Time	Amortized Time	Speedup
TFHE-rs	1	9.8ms	9.8ms	1.0
	2	13.6ms	6.8ms	1.4
This work	2048	5.1s	2.49ms	3.9
	4096	5.2s	1.26ms	7.8
	6144	6.8s	1.1ms	8.9

numbers in up to 4 times, but the additional cost for computing it makes it unlikely to be a practical choice. Instead, one could consider methods such as RLWE Conversion [CDKS21] and Partial LWE [BCL+24], which could reduce the packing key switching key to just a few megabytes. For comparison, key switching keys in TFHE-rs, which are used to reduce the dimension of the ciphertext, take 36.7, 65.2, 366.8, and 3055.3 MB, respectively, for 2, 4, 6, and 8-bit messages. Using the FFT-KS from [BCL+24] (based on those techniques), KS keys can be reduced to just 0.1, 2.8, 2.0, and 9.0MB, respectively. These gains cannot be directly mapped to the packing KS, but applying the equations from Table 2 to our parameters leads to key sizes of similar order of magnitude. Notwithstanding, one also needs to consider the impacts of the increased noise overhead introduced by RLWE conversion and other possible improvements from Partial LWE. We leave this assessment for future work.

7.3 Results for larger input dimension

Increasing the RLWE dimension N without changing other parameters could enable one to reduce the Hamming weight of the key while keeping the same security level. However, estimating security for these parameters can be hard, as tools such as the Lattice estimator do not provide reliable results. Notwithstanding, we present results for dimensions $N = 4096$ and $N = 8192$, aiming at characterizing the performance of our bootstrapping for larger dimensions. As we can't further optimize the Hamming weight, we choose $h = 32$ in all cases, as this value is broadly used in related literature [BKSS25]. Table 7 shows our results.

The results show that the concrete cost of our bootstrapping only grows logarithmic with the input dimension, enabling bootstrapping for large ciphertexts without the need of dimension-reduction key switchings that are often used in FHEW-like bootstrapping literature. The key size, while also asymptotically growing logarithmic with the input dimension, remains almost the same, being mostly defined by the Hamming Weight and choice of bootstrapping key (FBS_1 or FBS_2).

Table 7. Sparse amortized bootstrapping for ternary and arbitrary keys for larger input dimensions. The parameters $(h, \ell_A, q/\sigma) = (32, 19, 2^{24})$ are the same for all cases. The security level is estimated to be at least 133 bits.

Type	N	Keys	Bootstrap. Key Size	Prec. (bits)	Failure prob.	Total Time	Amortized Time
Tern.	4096	RPC_2+FBS_1	20.03MB	6	2^{-71}	20.53s	5.01ms
	8192	RPC_3+FBS_1	22.03MB	6	2^{-70}	45.43s	5.55ms
	4096	RPC_2+FBS_2	80.13MB	8	2^{-64}	130.67s	31.90ms
	8192	RPC_3+FBS_2	88.13MB	8	2^{-64}	286.55s	34.98ms
Arbi.	4096	RPC_2+FBS_1	84.00MB	4	2^{-73}	23.71s	5.79ms
	8192	RPC_3+FBS_1	86.00MB	4	2^{-73}	51.37s	6.27ms
	4096	RPC_2+FBS_2	1.08GB	6	2^{-72}	169.02s	41.26ms
	8192	RPC_3+FBS_2	1.09GB	6	2^{-73}	360.34s	43.99ms

7.4 Comparison with other works

Table 1, in the introduction, compares our solution with state-of-the-art works on bootstrapping with polynomial noise overhead. Besides them, we also compare with other approaches, remarking that different aspects must be considered for each of them.

Partial LWE Partial LWE [BCL+24] is a recently-introduced assumption (with direct reductions with standard ones) that is used to improve bootstrapping and KS performance in (non-amortized) TFHE. In [BCL+24], the 8-bit bootstrapping of TFHE-rs is improved from 415ms to 306ms. This result, however, is optimized for a failure probability (FP) of $< 2^{-13.9}$, which hinders a direct comparison with our work, since we focus on FP between 2^{-121} and 2^{-63}. By increasing our FP to 2^{-17}, we can bootstrap 8-bit messages in just 8.45ms (amortized for 2048 messages), which represents an improvement of 36.2 times over their result. These, however, should not be seen as alternative approaches, as Partial LWE could also be employed to accelerate our results. Specifically, notice that our FBS_2 key has 551.8 bits of security, which, with the use of Partial LWE, could be lowered to 128 bits in exchange for improving performance.

Methods with superpolynomial noise overhead Avoiding superpolynomial noise overhead is one of the main points of our work and, more generally, of the amortized bootstrapping line of research [MS18, GPvL23, LW23a, LW23b, DKMS24]. Nonetheless, it's unavoidable to wonder how the performance of these techniques compare in practice. The state of the art for superpolynomial methods is given by [BKSS25], which bootstraps 2, 4, 6, and 8-bit messages in 0.78, 1.57, 2.67, and 3.75ms, respectively. As one would expect, their results are better than ours, but we significantly narrow the gap compared to previous literature. In fact, for the first time, we have those methods at the same order of magnitude in terms

of amortized time (for messages of up to 6 bits), with the difference being as little as just 650 microseconds for 6-bit messages. Furthermore, latency-wise, our method is 1.6 times faster than [BKSS25] for 6-bit messages, as we bootstrap half the number of messages at a time.

Acknowledgments

This work was partly supported by the São Paulo Research Foundation (FAPESP), Brasil. Process Number 2023/12755-8, by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme in the scope of the CONFIDENTIAL6G project under Grant Agreement 101096435, and by Teaching, Research and Extension Support Fund (FAEPEX) under the Incentive Program for New Professors (PIND), process number 3389/23. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

ABD16. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178. Springer, Berlin, Heidelberg, August 2016.

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Berlin, Heidelberg, August 2014.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169 – 203, October 2015. Place: Berlin, Boston Publisher: De Gruyter.

BBL17. Daniel Benarroch, Zvika Brakerski, and Tancrède Lepoint. FHE over the integers: Decomposed and batched in the post-quantum regime. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 271–301. Springer, Berlin, Heidelberg, March 2017.

BCL+24. Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, Adeline Roux-Langlois, and Samuel Tap. New Secret Keys for Enhanced Performance in (T)FHE. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, pages 2547–2561, New York, NY, USA, December 2024. Association for Computing Machinery.

BDF18. Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 217–251. Springer, Cham, May 2018.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014.

BIP+22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 188–215. Springer, Cham, December 2022.

BKSS25. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping Small Integers With CKKS. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 330–360, Singapore, 2025. Springer Nature.

BLP+13. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

BST20. Florian Bourse, Olivier Sanders, and Jacques Traoré. Improved secure integer comparison via homomorphic encryption. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 391–416. Springer, Cham, February 2020.

BTH22. Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22International Conference on Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 521–541. Springer, Cham, June 2022.

CCP+24. Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks Against the IND-CPA$^{\mathrm{d}}$ Security of Exact FHE Schemes. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2505–2519, Salt Lake City UT USA, December 2024. ACM.

CDKS21. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21International Conference on Applied Cryptography and Network Security, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Cham, June 2021.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Berlin, Heidelberg, December 2016.

CGGI17. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Cham, December 2017.

CHK+17. Jung Hee Cheon, Kyoohyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. A practical post-quantum public-key cryptosystem based on spLWE. In Seokhie Hong and Jong Hwan Park, editors, *ICISC 16*, volume 10157 of *LNCS*, pages 51–74. Springer, Cham, November / December 2017.

CIM19. Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 106–126. Springer, Cham, March 2019.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Cham, December 2021.

DKMS24. Gabrielle De Micheli, Duhyeong Kim, Daniele Micciancio, and Adam Suhl. Faster amortized FHEW bootstrapping using ring automorphisms. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14604 of *LNCS*, pages 322–353. Springer, Cham, April 2024.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

DvW21. Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 3–32. Springer, Cham, December 2021.

GBA21. Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR TCHES*, 2021(2):229–253, 2021.

GBA24. Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized Software for FHE over the Torus. *Journal of Cryptographic Engineering*, July 2024.

Gen09. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

GPvL23. Antonio Guimarães, Hilder V. L. Pereira, and Barry van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 3–35. Springer, Singapore, December 2023.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

KF17. Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 3–26. Springer, Cham, April / May 2017.

LMK+23. Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 227–256. Springer, Cham, April 2023.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.

LW23a. Feng-Hao Liu and Han Wang. Batch bootstrapping I: A new framework for SIMD bootstrapping in polynomial modulus. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 321–352. Springer, Cham, April 2023.

LW23b. Feng-Hao Liu and Han Wang. Batch bootstrapping II: Bootstrapping in polynomial modulus only requires $\tilde{O}(1)$ FHE multiplications in amortization. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 353–384. Springer, Cham, April 2023.

LW23c. Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{\mathrm{O}}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 101–132. Springer, Singapore, December 2023.

MS18. Daniele Micciancio and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl, July 2018.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

SV14. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *DCC*, 71(1):57–81, 2014.

Zam22. Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs - commit b7d33e6b3f45b432af82637042214fa72bc7036c (Fri Mar 7 08:34:42 2025 +0100).

A Additional algorithms for packing key switching

Algorithm 7: Sparse packing key switching - key generation

Input : an LWE secret key $s \in \mathbb{Z}_q^N$ with Hamming weight h and its index set $\tilde{s} = \mathsf{idx}(s)$
Input : noise-performance trade-off parameters $k \in [1, h]$ and $B \leq N$ s.t. $B|N$
Output: two vectors V and C, s.t. $\tilde{s} \subset \{(iB \mod N) + C_i : i \in [0, kN/B)\}$, $V_i = s_{(iB \mod N)+C_i}$, and $C_i < B$, for all $i \in [0, kN/B)$, or $\perp$ if s cannot be represented in this format

$V \leftarrow \{0\}$; $C \leftarrow \{0\}$
for $j \leftarrow 0$ **to** N/B **do**
  $i \leftarrow 0$
  for $t \leftarrow 0$ **to** B **do**
    if $s_{jB+t} \neq 0$ **then**
      $(V_{jk+i}, C_{jk+i}, s_{jB+t}, i) \leftarrow (s_{jB+t}, t, 0, i+1)$
  if $i > k$ **then**
    Return $\perp$
if $s \neq \{0\}$ **then**
  Return $\perp$
Return (V, C)

B Average-case noise analysis

To improve parameters in practice, we consider approximate gadget decompositions [CGGI16] and performance an average case noise analysis of our method. Most of this analysis comes straightforwardly from applying previous literature [CGGI16, CGGI16, BST20, GBA21] to the equations of Sections 3 to 6. In particular, our constructions requires three main building blocks: GSW external products, key switchings (for the automorphisms), and CMUX's. We have average-case noise analysis for all of them directly from [CGGI16,CGGI17]. They do not consider the impact of sparse secrets in their analysis, but several other follow-up works have extended their analysis to consider it [GBA21, CCP+24]. Based on them, we obtain the following equations. For all of them, consider:

- ℓ_{A} and β are parameters of the approximate gadget decomposition of the RGSW or ksk input.
- N is the RLWE dimension and k is the MLWE rank ($k = 1$ for us in all cases)
- σ_a and μ_a are, respectively, the noise standard deviation and the encrypted message of some ciphertext "a"
- h is the hamming weight of the key of the RGSW or ksk input.

RGSW external product. Given an RGSW ciphertext A and a RLWE ciphertext b, it produces an RLWE ciphertext c with the following noise variance:

$$\sigma_{\mathsf{out}}^2 < (k+1)\ell_{\mathsf{A}} N \left(\frac{\beta}{2}\right)^2 \sigma_A^2 + (1+kh)\,\|\mu_A\|_2^2\,\frac{1}{12}\left(\frac{q}{\beta^{\ell_{\mathsf{A}}}}\right)^2 + \|\mu_A\|_2^2\,\sigma_b^2 \qquad (5)$$

CMUX Given an RGSW ciphertext C and two RLWE ciphertexts a and b, it produces an RLWE ciphertext c with the following noise variance:

$$\sigma^2_{\mathsf{out}} < (k+1)\ell_{\mathsf{A}} N \left(\frac{\beta}{2}\right)^2 \sigma^2_C + (1+kh)\frac{1}{12}\left(\frac{q}{\beta^{\ell_{\mathsf{A}}}}\right)^2 + \mathsf{max}(\sigma^2_a, \sigma^2_b) \tag{6}$$

Key switching. Given an RLWE ciphertext a and a set of RLWE ciphertexts ksk, it produces an RLWE ciphertext b with the following noise variance:

$$\sigma^2_{\mathsf{out}} < k\ell_{\mathsf{A}} N \left(\frac{\beta}{2}\right)^2 \sigma^2_{\mathsf{ksk}} + (kh)\frac{1}{12}\left(\frac{q}{\beta^{\ell_{\mathsf{A}}}}\right)^2 + \sigma^2_a \tag{7}$$

C Estimating δ

We estimate the value of δ by generating a large number of random keys and measuring the value of B that would be required for each of them. Specifically, for each pair of parameters (N, h), we generated 100 thousand keys, computed the required value of B for each, and built a histogram. From the histogram, we computed the cumulative distribution and generated Table 8. To give a concrete example, for $N = 2048, h = 17$, from the 100 thousand keys: None of them would work with $B = 2^7$; $1/2^{4.21} = 5.4\%$ of them would work with $B = 2^8$; $1/2^{0.19} = 87.6\%$ of them would work with $B = 2^9$; and all would work with $B = 2^{10}$.

Table 8. Estimated loss of security (δ) for $N = 2048$

h	$B = 2^7$	$B = 2^8$	$B = 2^9$	$B = 2^{10}$
17	-	4.21	0.19	0
18	-	3.71	0.15	0
19	-	3.29	0.12	0
20	16.61	2.95	0.09	0
21	-	2.62	0.07	0
22	16.61	2.32	0.06	0
23	-	2.05	0.05	0
24	16.61	1.83	0.03	0
25	16.61	1.64	0.03	0
26	16.61	1.45	0.02	0
27	15.61	1.29	0.02	0
28	14.29	1.16	0.01	0
29	13.61	1.02	0.01	0
30	13.02	0.91	0.01	0
31	12.7	0.81	0.01	0
32	11.44	0.72	0	0
33	10.68	0.64	0	0