Parameter Optimization and Larger Precision for (T)FHE

Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, Samuel Tap

Zama, Paris, France - https://zama.ai/
{loris.bergerat, anas.boudi, quentin.bourgerie, ilaria.chillotti, damien.ligier, jb.orfila, samuel.tap}@zama.ai

Abstract

In theory, Fully Homomorphic Encryption schemes allow users to compute any operation over encrypted data. However in practice, one of the major difficulties lies into determining secure cryptographic parameters that minimize the computational cost of evaluating a circuit. In this paper, we propose a solution to solve this open problem. Even though it mainly focuses on TFHE, the method is generic enough to be adapted to all the current FHE schemes.

TFHE is particularly suited, for small precision messages, from Boolean to 5-bit integers. It is possible to instantiate bigger integers with this scheme, however the computational cost quickly becomes unpractical.

By studying the parameter optimization problem for TFHE, we observed that if one wants to evaluate operations on larger integers, the best way to do it is by encrypting the message into several ciphertexts, instead of considering bigger parameters for a single ciphertext.

In the literature, one can find some constructions going in that direction, which are mainly based on radix and CRT representations of the message. However, they still present some limitations, such as inefficient algorithms to evaluate generic homomorphic lookup tables and no solution to work with arbitrary modulus for the message space.

We overcome these limitations by proposing two new ways to evaluate homomorphic modular reductions for any modulo in the radix approach, by introducing on the one hand a new hybrid representation, and on the other hand by exploiting a new efficient algorithm to evaluate generic lookup tables on several ciphertexts. The latter is not only a programmable bootstrapping but does not require any padding bit, as needed in the original TFHE bootstrapping. We additionally provide benchmarks to support our results in practice.

Finally, we formalize the parameter selection as an optimization problem, and we introduce a framework based on it enabling easy and efficient translation of an arithmetic circuit into an FHE graph of operation along with its

optimal set of cryptographic parameters. This framework offers a plethora of features: fair comparisons between FHE operators, study of contexts that are favorable to a given FHE strategy/algorithm, failure probability selection for the entire use case, and so on.

Contents

1 Introduction

Fully Homomorphic Encryption (FHE) refers to an encryption scheme that allows to perform a potentially unlimited amount of computation over encrypted data. FHE schemes have attracted a lot of attention in the last decade. Indeed, they may solve many real world applications on which the privacy of some manipulated data has to be preserved.

The constructions that are mainly studied nowadays are based on hard problems over lattices: LWE [Reg05], and its variant RLWE [LPR10, SSTX09]. The (R)LWE-based schemes mainly used nowadays are BGV [BGV12], B/FV [Bra12, FV12], HEAAN [CKKS17], GSW [GSW13], FHEW [DM15] and TFHE [CGGI20].

While the constructions have seen a large improvement in the last decade, by proposing operations that are more and more efficient, one of the main problems of FHE schemes remains to *find good cryptographic parameters*. Such parameters need to both be secure and make the operations as efficient as possible, in terms of either computational cost, memory or power consumption. Solving this problem is fundamental if we plan for a large scale adoption of FHE schemes. Regarding the security constraints, the LWE/Lattice-estimator [APS15] is the main tool to evaluate the security of parameters for an LWE-based cryptosystem[1]. However, it does not help finding efficient parameters for a given use-case. *Finding the optimal parameter set is even harder and no solution has been proposed yet.*

In this paper, we mainly focus on TFHE scheme [CGGI20]. This scheme is particularly interesting because it offers a bootstrapping technique that is able to reduce the noise of ciphertexts and, at the same time, to evaluate a function on the input, expressed as a Look-Up Table (LUT). This is often called programmable bootstrapping (PBS). Unfortunately, in practice, the bootstrapping takes as input a single LWE ciphertext encrypting a small message (say at most 8 bits). To evaluate operations on large precision messages, the only known way is to split the message into many ciphertexts and then build the circuit as a combination of linear operations and PBSs. The solutions presented until now, however, are not very efficient.

State of the art

In the literature, a few compilers [DKS+20][GKT22] for FHE schemes have been proposed: they mainly optimize the circuit to make it as FHE friendly as possible. Most of them target schemes like HEAAN in [CKKS17] where there is no real bootstrapping algorithm and which has batching and SIMD capabilities that TFHE does not have. The noise management for HEAAN is quite different than the noise management for TFHE, the former providing approximate results. For schemes such as BGV, B/FV or HEAAN, that have the tendency to avoid bootstrapping in favor of leveled operations, the existing compilers perform some kind of parameter selection.

[1] In what follows, the security has been estimated with the commit made on January 5, 2023: https://github.com/malb/lattice-estimator/tree/f9f4b3c69d5be6df2c16243e8b1faa80703f020c

For these schemes, there exists a line of work aiming to reduce the multiplicative depth of the circuit to be homomorphically evaluated [CAS17, ACS20, LLOY20]. This is due to the fact that these schemes are parametrized with a fixed number of levels, and every multiplication consumes one of the levels. Once all the levels are consumed, no more multiplications can be performed and decryption or bootstrapping is required. This is not an approach used in TFHE-like schemes, where the multiplicative depth is not a measure taken into account, since the non linear operations are performed by using a bootstrapping.

For TFHE-like schemes, the circuits evaluated by these compilers [CMG$^+$18, CDS15] are binary circuits, using gate bootstrapping, with hard-coded parameters and focus on optimizing the Boolean circuit instead.

Some other existing works tried to achieve a similar objective as our contribution, but with a reduced scope. As instance, in [MML$^+$22] the authors tried to improve the parameter generation for BGV, and in [Kle22], Klemsa proposes an approach to automatize the setup of parameters for TFHE with particular attention in efficiently using resources during the bootstrapping step (e.g. size of bootstrapping keys). The task we try to solve is wider and aims to provide a generic approach to automatically select the best parameters according to a given cost model for an arbitrary graph of FHE operators while guaranteeing correctness and security.

We suggest this paper [VJH21] for more information and for comparisons on all existing FHE compilers. To the best of our knowledge, no one has ever presented a result on optimization of parameters for an FHE scheme including a bootstrapping with a flexibility for multi-precision plaintexts.

In the state of the art, several approaches using many ciphertexts to represent a single message are proposed. We can summarize these approaches in two main categories: the radix and the CRT (Chinese Reminder Theorem) representations.

The radix representation consists in decomposing a message into several chunks according to a decomposition base. It is very similar to the representation in base 10 we use in our daily lives, where to represent a large number we use several digits. Then the idea is to put each of the elements of the decomposition into a separate ciphertext and to define the new encryption of the large message as the list of these ciphertexts.

The CRT approach consists in representing a number x modulo a large integer $\Omega = \prod_{i=0}^{\kappa} \omega_i$, where the ω_i are all co-primes, as the list of its residues $x_i = x \mod \omega_i$. Each of the reduced elements is then encrypted into a different ciphertext and, as for the radix approach, the new encryption of the large message modulo Ω is the list of these ciphertexts. Observe that the CRT approach in the plaintext space is different from the well known SIMD style [GHS12].

In order to use these two approaches in TFHE, the elements of the decomposition (for the radix approach) and the residues (for the CRT approach) need to be quite small (generally less than 8 bits).

The approach of splitting a message into multiple ciphertexts has already been proposed for binary radix decomposition in FHEW [DM15] and TFHE [CGGI20],

and for other representations in [BST20], [GBA21], [KO22], [CZB+22], [LMP21] and [CLOT21]. However, none of them takes advantage of carry buffers to make the computations more efficient between multi-ciphertext encrypted integers by avoiding bootstrapping as much as possible. In [GBA21] they propose two approaches to evaluate the PBS over these multi-ciphertexts inputs, called *tree-based* and *chained-based* approaches (that we shorten by Tree-PBS and Chained-PBS). The Chained-PBS method is generalized in [CZB+22] to any function in exchange of a larger plaintext space. In [CLOT21], the authors propose for the first time a WoP-PBS technique, i.e., a PBS that does not require a bit of padding. After them, two different WoP-PBS were proposed in [LMP21] and [KS21].

The idea of using the CRT approach is mentioned in [KS21] but unfortunately no details are provided. The authors do not change the traditional TFHE encoding to fit the CRT representation. In our paper, we provide detailed algorithms to describe the use of the CRT in the plaintext space with two different approaches (with or without carry buffers, along with their respective encoding). Concerning bootstrapping, they describe how to trivially construct polynomials for the blind rotation, which allows them to only evaluate a narrow set of functions (every CRT element mapped to an output CRT element).

These techniques are the first step towards larger precision. Indeed, they have some limitations. In particular, the radix approach does not allow any modulo to be represented, but only multiples of a certain base (or bases), and the CRT is limited on the maximal number that could be represented, because there exists a very limited amount of primes or co-primes smaller than 8 bits.

While arithmetic operations can be evaluated quite straight forwardly for these representations, the bootstrapping and generic LUT evaluation is very inefficient, and the only known technique is the Tree-PBS proposed by [GBA21]. This technique becomes very inefficient as the number of ciphertexts encrypting a single large message increases.

Our contributions

The problem of finding the optimal set of cryptographic parameters for a homomorphic evaluation is still an open problem, and represents an obstacle in the path towards large-scale adoption of FHE.

In this paper, we introduce the first optimization procedure for TFHE-like schemes. The idea is to exploit FHE knowledge to build a good optimization model and to speed up the process with TFHE related shortcuts. A noise formula and a cost model are attached to each of the FHE operators, in order to quantify its impact on the noise growth and on the execution time. In the end, we succeed in translating the FHE optimization problem into a more classical optimization problem (simplified along the way), where already known and powerful optimization techniques finally take over, such as the branch-and-bound algorithm.

By finding the fastest set of parameters for different contexts, one can truly compare FHE operators together. Some of the comparisons we make in this paper

put some light on the relation between the precision of the encrypted integers, and the time needed to compute over them. In particular, it is clear that if we want to work with large precision in TFHE, it is more efficient to use several ciphertexts to encrypt a single message, instead of considering huge parameters so it fits into a single one. TFHE ciphertexts are in fact *limited to a precision of 8 bits at most*: above that, computation is considered too slow.

In this paper we overcome some limitations that were still present in such constructions. For instance, we add two new techniques in order to compute a generic modular reduction for any modulo in the radix strategy, which was an open problem. We also introduce an *hybrid approach* mixing CRT and radix representations in order to get the best of both worlds and having the possibility to implement any homomorphic modular integer arithmetic. We also introduce a new algorithm to compute a bootstrapping on one or several ciphertexts, which do not need to have a known bit of padding, and allows us to evaluate at once a generic lookup-table on a large integer. Using this new algorithm, it is then possible to evaluate in an efficient manner a generic lookup-table on radix, CRT, or hybrid integer representation, which was not possible until this paper. We also propose *benchmarks* proving the practicality of our new techniques.

Finally, by formalizing the study done on optimization, we design the *first optimization framework* for FHE computation. Roughly speaking, it is a generic approach taking as input a graph of mathematical operators, such as additions, multiplications or LUT evaluations, and a list of *translation rules*, i.e., various ways to transform these clear operations into FHE operations. Numerical values across this graph are associated with some metadata regarding their precision and whether or not it should be encrypted. In a nutshell, the output provided by the optimization framework is an *optimal graph* of FHE operators along with the *optimal parameter set* for this graph.

As already mentioned above, this optimization framework allows to compare different FHE algorithms together. Indeed, within the same use-case, one can find the best parameters for different combinations of FHE operators (computing the same plaintext function), in a fair way. As a matter of fact, it will take into account the probability of failure as well as the output noise so they are the same for all tested combinations. We also show how to optimize a global probability of failure for the entire use-case. We demonstrate that adding more bootstrapping can speed up some homomorphic computations and we prove that the position of a key switch operator has a non negligible impact on the efficiency of a circuit. Finally we describe ways to use our framework to take into account other constraints such as having a consensus-friendly FHE evaluation, or allowing an optimization for more than a single pair of bootstrapping key and key switching key.

Paper organization

A background on probability and FHE, as well as more detailed techniques for homomorphic arithmetic encoded on single/multiple ciphertexts, are provided in section 2.

Section 3 introduces one of this paper's contribution: a process to select cryptographic parameters. It starts by formalizing it as an optimization problem, and details simplifications one can do to speed the optimization process.

Section 4 gives details about another contribution: large-precision homomorphic modular integers for any modulus, thanks to two new homomorphic modular reduction algorithms, and hybrid representation bridging the gap between the CRT and radix representations. Finally we describe in this section a new algorithm to compute homomorphic lookup-tables over multiple LWE ciphertexts and how to leverage it to compute any function on our large-precision homomorphic modular integers. We conclude this section with some comparisons with the state of the art and benchmarks.

Section 5 provides a description of our optimization framework with its many features. Indeed to get the best out of our modular integers, we need to find the best parameter set which is done with the optimization framework.

Section 6 presents some conclusive remarks, open problems and future works.

2 Preliminaries

Notations and probability background.

In the rest of the paper, we use the notation $\mathcal{N}(\mu, \sigma^2)$ to indicate a normal distribution with mean μ and variance σ^2. We note by Var and $\mathbb{E}$ the variance and expectation of a probability distribution respectively. We note by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the sets of natural, integer and real numbers respectively. We often use the shortcut $\mathbb{Z}_\Omega$ to represent $\mathbb{Z}/\Omega\mathbb{Z}$. We note $\mathfrak{R} = \mathbb{Z}[X]/(X^N + 1)$, where N is a power of 2, and $\mathfrak{R}_\Omega = \mathbb{Z}_\Omega[X]/(X^N + 1)$. We note by $\mathbb{P}$ a probability.

Definition 1 (Standard score) *Let $A \hookleftarrow \mathcal{N}(0, \sigma^2)$ (centered normal distribution), let p_{fail} be a failure probability and let erf be the error function $\mathsf{erf}(z) \mapsto \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \mathsf{dt}$. We define the* standard score *z^* for p_{fail} as $z^*(p_{\mathsf{fail}}) = \sqrt{2} \cdot \mathsf{erf}^{-1}(1 - p_{\mathsf{fail}})$ and we have: $\mathbb{P}(A \notin]-z^*\sigma, z^*\sigma[) \leq p_{\mathsf{fail}}$.*

Let $t \in \mathbb{R}$, we have $z^*(p_{\mathsf{fail}}) \cdot \sigma \leq t \Rightarrow \mathbb{P}(A \notin]-t, t[) \leq p_{\mathsf{fail}}$

2.1 FHE Background

The security of many FHE schemes is based on the hardness of the LWE problem and its variants. There are several types of ciphertexts involved in TFHE. We start by defining the encoding function that we use in this entire paper for GLWE ciphertexts and later we recall the definition of GLWE ciphertexts including its special cases of RLWE and LWE ciphertexts.

Definition 2 (GLWE Encode & Decode) *Let $q \in \mathbb{N}$ be a ciphertext modulus, and let $p \in \mathbb{N}$ a message modulus, and $\pi \in \mathbb{N}$ the number of bits of padding[2]. We have $2^\pi \cdot p \leq q$ and $2^\pi \cdot p$ is the plaintext modulus. Let $M \in \mathfrak{R}_p$ be a message. We define the encoding of M as: $\widetilde{M} = \mathsf{Encode}(M, 2^\pi \cdot p, q) = \lfloor \Delta \cdot M \rceil \in \mathfrak{R}_q$ with $\Delta = \frac{q}{2^\pi \cdot p} \in \mathbb{Q}$ the scaling factor (see a visual example in Figure 1). To decode, we compute the following function: $M = \mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) = \left\lfloor \frac{\widetilde{M}}{\Delta} \right\rceil \in \mathbb{Z}_{2^\pi \cdot p}$.*

In practice $\widetilde{M}$ contains a "small" error term $E = \sum_{i=0}^{N-1} e_i \cdot X^i \in \mathbb{Q}[X]/(X^N+1)$, so we can rewrite $\widetilde{M} = \Delta \cdot M + E \in \mathbb{Z}_q$. The decoding algorithm fails if and only if there is at least one $i \in [\![0, N-1]\!]$ such that $|e_i| \geq \frac{\Delta}{2}$. We can note this probability as follow:

$$\mathbb{P}\left(\bigcup |e_i| \geq \frac{\Delta}{2}\right) = \mathbb{P}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right) \tag{1}$$

Definition 3 (GLWE Ciphertext) *Given an encoding $\widetilde{M} \in \mathfrak{R}_q$ and a secret key $\vec{S} = (S_1, \cdots, S_k) \in \mathfrak{R}_q^k$, with coefficients either sampled from a uniform binary,*

[2]For simplicity we use a power of 2 for the padding, but this is not a necessary condition.

uniform ternary or Gaussian distribution, a GLWE ciphertext of $\widetilde{M}$ *under the secret key* $\vec{S}$ *is defined as the tuple:*

$$\mathsf{CT} = \left(A_1, \cdots, A_k, B = \sum_{i=1}^{k} A_i \cdot S_i + \widetilde{M} + E \right) \in \mathsf{GLWE}_{\vec{S}}(\widetilde{M}) \subseteq \mathfrak{R}_q^{k+1}$$

such that $\{A_i\}_{i=1}^k$ *are polynomials in* $\mathfrak{R}_q$ *with coefficients sampled from the uniform distribution in* $\mathbb{Z}_q$*,* E *is a noise (error) polynomial in* $\mathfrak{R}_q$*, with coefficients sampled from a Gaussian distributions* χ_σ*, and with* $\widetilde{M} = \mathsf{Encode}\,(M, p, q)$*. The parameter* $k \in \mathbb{Z}_{>0}$ *represents the number of polynomials in the GLWE secret key.*
A GLWE ciphertext with $N = 1$ *is called* LWE ciphertext*: in this case we note the size of the secret key by* $n = k$*, and we note both the ciphertext and the secret with a lower case, e.g.* ct *and* $\vec{s}$*. A GLWE ciphertext with* $k = 1$ *and* $N > 1$ *is called* RLWE ciphertext*.*

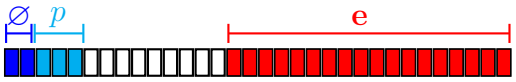

Figure 1: Plaintext binary representation with $p = 8 = 2^3$ (cyan), $\pi = 2$ (dark blue) such that $2^\pi \cdot p \leq q$, the error e (red). The white part is empty. The MSB are on the left and the LSB on the right.

In TFHE-like schemes, another type of ciphertext is used, and it is called GGSW (Generalized GSW [GSW13]). A GGSW ciphertext is composed of $(k+1)\ell$ GLWE ciphertexts, encrypting the same message times elements of the secret key with some redundancy. The redundancy is defined by a decomposition base β and a number of levels ℓ in the decomposition. GGSW ciphertexts are used for bootstrapping keys and in the circuit bootstrapping, later described.

Key Switching. An *LWE-to-LWE key switching* is a homomorphic operator to change the secret key as well as a few parameters, and details can be found in [CGGI20, CLOT21]. It takes as input an LWE ciphertext $\mathsf{ct_{in}} \in \mathsf{LWE}_{\vec{s}'}(\tilde{m})$ encrypting an encoding $\tilde{m}$ of a message m under a secret key $\vec{s}' \in \mathbb{Z}^{n'}$, and a key switching key KSK encrypting $\vec{s}'$ with redundancy under another LWE secret key $\vec{s} \in \mathbb{Z}^n$. It returns an LWE ciphertext $\mathsf{ct_{out}} \in \mathsf{LWE}_{\vec{s}}(\tilde{m})$ encrypting m under the secret key $\vec{s}$, with a larger noise. Its signature is $\mathsf{ct_{out}} \leftarrow \mathsf{KS}(\mathsf{ct_{in}}, \mathsf{KSK})$.

Bootstrapping. In FHE schemes, the technique used to reduce the noise is called *bootstrapping*. In TFHE-like schemes, bootstrapping is also able to evaluate a LUT at the same time. For this reason it is often called *programmable bootstrapping* [CGGI20, CJL+20, CJP21], or *PBS* in short. The PBS is composed of 3 sequential operators: a modulus switching (MS), a blind rotation (BR) and a sample extraction (SE). A PBS, takes as input an LWE ciphertext $\mathsf{ct_{in}} \in \mathsf{LWE}_{\vec{s}}(\tilde{m})$ encrypting an encoding $\tilde{m}$ of a message m under a secret uniform binary key $\vec{s} \in \mathbb{Z}^n$,

a bootstrapping key BSK encrypting the bits of $\vec{s}$ as GGSW ciphertexts under a secret key $\vec{S'} \in \mathfrak{R}^k$, and a polynomial P_L encoding a r-redundant[3] LUT for $x \mapsto L[x]$. The PBS returns an LWE ciphertext $\mathsf{ct_{out}}$ encrypting $L[m]$ under the secret key $\vec{s'}$, extracted from $\vec{S'}$, with smaller noise (if the parameters are chosen appropriately and if the input LWE ciphertext did not contain too much noise). There is however a probability of failure where the output is actually $L[m + \epsilon]$ with $\epsilon \neq 0$. The signature of the PBS is: $\mathsf{ct_{out}} \leftarrow \mathsf{PBS}(\mathsf{ct_{in}}, \mathsf{BSK}, P_L)$. To simplify notation, we note $\mathsf{ct_{out}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct_{in}}, \mathsf{PUB}, P_f)$ where $\mathsf{PUB} = (\mathsf{BSK}, \mathsf{KSK})$ when a KS is followed by a PBS.

Two additional parameters can be used in the PBS to obtain a generalized PBS as in [CLOT21]: $(\varkappa, \vartheta)$. These two parameters define the exact part of the plaintext that is extracted by modulus switching during the PBS.

LUT evaluation. In 2017, Chillotti et al. [CGGI20] proposed an operator called *circuit bootstrapping*, transforming an LWE ciphertext into a GGSW ciphertext. It consists in performing some PBS followed by an LWE-to-GLWE functional KS. The later operator converts an LWE ciphertext encrypting m into a GLWE ciphertext encrypting the constant polynomial m. The authors also proposed two operators to evaluate LUTs in a leveled way, called *horizontal and vertical packing*. They both take as input a d-bit message msg, encrypted as a list of d GGSW ciphertexts encrypting one of its bits. They also take in input α LUTs $L_0 = [l_{0,0}, \cdots, l_{0,2^d-1}], \ldots L_{\alpha-1} = [l_{\alpha-1,0}, \cdots, l_{\alpha-1,2^d-1}]$: the goal is to compute the result of the evaluation of the LUTs on the input message, i.e., return encryptions of $l_{0,\mathsf{msg}}, \ldots, l_{\alpha-1,\mathsf{msg}}$. Both operators use CMux gates, either as a tree or in a blind rotation, to compute the LWE result (that can be a GLWE in horizontal packing). Horizontal packing is interesting when many LUTs should be evaluated in parallel, while vertical packing is interesting when a single (large) LUT needs to be evaluated. They are two extremes of a trade-off for the evaluation of homomorphic LUTs and a mixed solution has been proposed generalizing both of them.

2.2 Modular Arithmetic with a Single LWE Ciphertext

In this section we adapt the encoding proposed in definition 2 in order to include a *carry space* into the plaintext space. The core idea is to give enough room in a ciphertext encrypting an integer message modulo $\beta \in \mathbb{N}$ to store more than just the message but also potential carries coming from leveled operations such as addition or multiplication with a known integer. In order to keep track of the worst case message in each ciphertext – i.e., check if there is still room to perform more operations – we use a metadata that we call *degree of fullness*.

[3]The redundancy of a LUT consists in repeating r times (r is a parameter depending on N and on the number of possible messages) the entries $L(i)$ of the LUT in the polynomial P_L. The redundancy is necessary to perform the rounding operation during bootstrapping [CLOT21, Section 2].

In practice, we split the traditional plaintext space into three different parts: the *message subspace* storing an integer modulo $\beta \in \mathbb{Z}$ (we call β the base), the *carry subspace* containing information overlapping β, and a bit of padding (or more) often needed for bootstrapping. In this context, we refer to the *carry-message modulo* as the subspace including both the message subspace plus the carry subspace, and we note it $p \in \mathbb{N}$. Figure 2 shows a visual example.

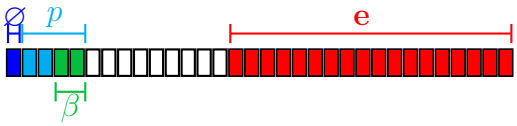

Figure 2: Plaintext binary representation with a base $\beta = 4 = 2^2$ (green), a carry subspace (cyan), a carry-message modulo $p = 16 = 2^{2+2}$ (cyan+green) such that $0 < \beta < p$, the error e (red), and a bit of padding is displayed in the MSB (dark blue). The white part is empty. So the plaintext modulo is $32 = 2^{2+2+1}$. This means that we have 2 bits in the carry subspace (set to 0 in a fresh ciphertext), that will contain useful data when one computes leveled operations.

The *degree of fullness*, that we note deg, of an LWE ciphertext ct encrypting a message $0 \leq m < p$, is equal to $\deg(\mathsf{ct}) = \frac{\mu}{p-1} \in \mathbb{Q}$, where μ is the known worst case for m, i.e., the biggest integer that m can be, such that $0 \leq m \leq \mu < p$. To ensure correctness, the degree of fullness should always be a quantity included between 0 and 1, where $\deg(\mathsf{ct}) = 1$ means that the carry-message subspace is full in the worst case.

We take advantage of the carry subspace to compute leveled operations and to avoid bootstrapping. In practice, *the carry subspace acts as a buffer* to contain the carry information derived from homomorphic operations and the degree of fullness acts as a measure that indicates when the buffer cannot support additional operations: once this limit is reached the carry subspace is emptied by bootstrapping. To be able to perform a leveled operation between two LWE ciphertexts of that type, they need to have the same base β, carry-message p and ciphertext modulus q. We now list the operators one can compute over such encrypted integers:

- *homomorphic addition* between two encrypted integers;
- *multiplication by a small integer constant*;
- *homomorphic opposite*, requiring a correction term;
- *homomorphic subtraction*, composed as an opposite and an addition;
- *homomorphic univariate function evaluation*, computed with PBS;
- *homomorphic multivariate function evaluation*, computed by using a trick that was already proposed in [CZB+22]. If the degrees of the ciphertexts allow, the idea is to concatenate two messages m_1 and m_2 (or more) respectively

encrypted in ct_1 and ct_2 by re-scaling the first one with constant multiplication to $\mu_2 + 1$ (where μ_2 is the worst possible value that can be reached by the m_2) and add it to ct_2 and finally compute a PBS on the concatenation. Once the two messages are concatenated in a single ciphertext, the bi-variate LUT L can be simply evaluated as a univariate LUT L' on the concatenation of m_1 and m_2. A visual example is proposed in Figure 3;

- *homomorphic multiplication* between two ciphertexts, computed by using the multivariate approach just described, for both the LSB multiplication (i.e. $m_1 \cdot m_2 \mod \beta$) and the MSB multiplication (i.e. $\left\lfloor \frac{m_1 \cdot m_2}{\beta} \right\rfloor$). If instead we want to compute the multiplication without any modular reduction, we can use well known techniques in TFHE literature such as in [CJL+20];
- *homomorphic carry/message extraction*, computed through PBS.

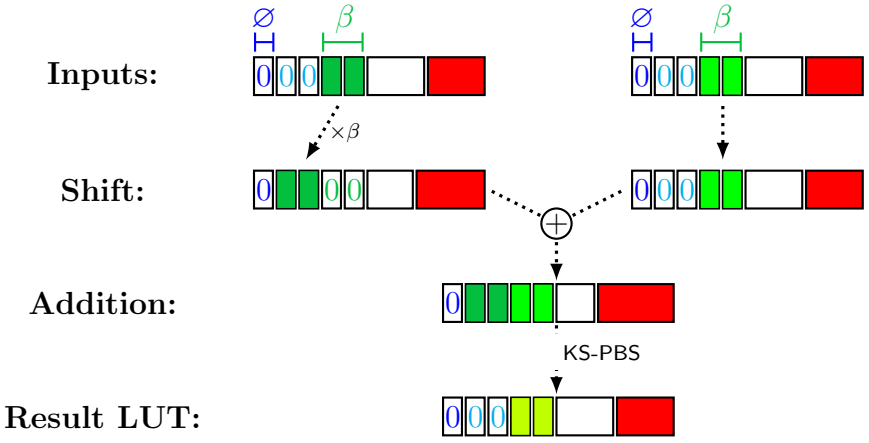

Figure 3: Example of a bi-variate LUT evaluation with shift and PBS.

We provide more details on these operations in Supplementary Material A.

Remark 1 *Generally with FHE one has to monitor noise growth. However, in this paper, we chose parameters such that the noise is always under a certain level if the degree of fullness has not reached the maximal value allowed. When we approach this value for the degree, a bootstrapping operation is performed and the noise is reduced at the same time. We give more details in Section 4.3.1.*

In the next paragraph, we provide details on how to build a LUT and evaluate the PBS on ciphertexts having p different from a power of 2.

PBS with p Not a Power of Two. No details, nor analysis have been provided yet in the literature about computing PBS when the plaintext space is not a power of two. We bring some clarity to this question in this paragraph.

When one wants to compute a traditional PBS evaluating a non-negacyclic function, it is required to have a bit of padding, which forces the plaintext space to be even. However the algorithm works the same with an odd p, the only difference lies in the way the r-redundant LUT is built. This also brings a slight modification in the evaluation of the error probability when computing such PBS.

Recall that such LUT encoded in the polynomial plaintext $\tilde{L} = \mathsf{Encode}(L, p', q)$ of a GLWE ciphertext and $\tilde{L}$ contains redundancy. We call *mega-cases* each block of successive redundant values. In a generic manner, if the LUT we want to compute is defined as $L : \mathbb{Z}_p \to \mathbb{Z}_{p'}$, $x \mapsto y_x$, we define the polynomial $\tilde{L}$ as:

$$\tilde{L} = X^{-\left\lfloor \frac{N}{2 \cdot p} \right\rceil} \cdot \left(\sum_{i=0}^{N-1} \left\lfloor \frac{q}{p'} \cdot y_{\left\lfloor \frac{i \cdot p}{N} \right\rfloor} \right\rceil \cdot X^i \right)$$

Proof 1 (Sketch) *With such a LUT, and p not a power of 2, we end up with two possible sizes for the mega-cases of the r-redundant LUT: either $\left\lfloor \frac{N}{p} \right\rfloor$ or $\left\lceil \frac{N}{p} \right\rceil$. For the correctness study, we will take the worst case scenario, i.e., considering $\left\lfloor \frac{N}{p} \right\rfloor$. The encoding function (definition 2) enables to have messages centered in the mega-cases when it comes to PBS, it means that the probability of going into the wrong mega-case during a PBS in the worst case scenario is when the error e_{MS} is bigger in absolute value than $\left\lfloor \frac{N}{p} \right\rfloor$ where e_{MS} is the error in the PBS after the modulus switch and before the blind rotation. It is easy to estimate e_{MS} as a variance, thanks to noise formulae. Since it is close to a Gaussian distribution, we can use a confidence interval to infer the probability to get into the wrong mega-case.* □

2.3 Modular Arithmetic with Several LWE ciphertexts

In TFHE, a single ciphertext can efficiently encrypt up to 8-bits of information. Larger messages should be encrypted in a different way: a possibility is to use many ciphertexts to encrypt a single large precision message in LWE. In that case, there are two options that are already used in the literature: the *radix representation* or the *CRT representation*. They are both valid approaches but have some limitations in their actual state. We briefly describe the two approaches and the limitations that we overcome in Section 4.

2.3.1 Radix-based large integers

The radix based approach consists in encrypting a large integer modulo $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$ as a list of $\kappa \in \mathbb{N}$ LWE ciphertexts. Each of the κ ciphertexts is defined according to a pair $(\beta_i, p_i) \in \mathbb{N}^2$ of parameters, such that $2 \leq \beta_i \leq p_i < q$, which respectively corresponds to the message subspace and the carry-message subspace involved with the modular arithmetic, as described in Section 2.2. Figure 4 gives a visual representation out of a toy example.

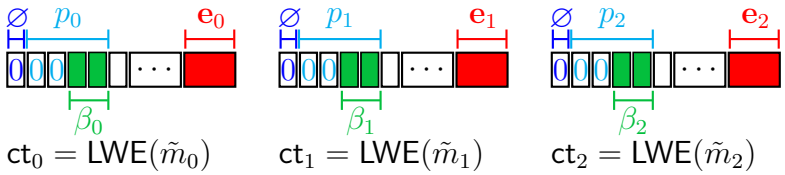

Figure 4: Plaintext representation of a fresh radix-based modular integer of length $\kappa = 3$ working modulo $\Omega = (2^2)^3$ with $\mathsf{msg} = m_0 + m_1 \cdot \beta_0 + m_2 \cdot \beta_0 \cdot \beta_1$. The symbol $\varnothing$ represents the padding bit needed for the PBS. For each block we have $\tilde{m}_i = \mathsf{Encode}\,(m_i, p_i, q)$. For all $0 \leq i < \kappa$ we have $\beta_i = 4$, $p_i = 16$, $\kappa = 3$ and $\Omega = 4^3$.

In practice, the restriction for Ω is that it has to be a product of small basis. Indeed, TFHE-like schemes do no scale well when one is increasing the precision, so the good practice is to keep $p_i \leq 2^8$.

To *encode* a message $\mathsf{msg} \in \mathbb{Z}_\Omega$, one needs to decompose it into a list of $\{m_i\}_{i=0}^{\kappa-1}$ such that $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right)$. Then we can independently call the Encode function (definition 2) on each m_i so we have $\tilde{m}_i = \mathsf{Encode}\,(m_i, 2^\pi \cdot p_i, q)$ with π the number of bits of padding. Finally we can encrypt each $\tilde{m}_i$ into an LWE ciphertext. To *decode*, we simply recompose the integer from the m_i values.

In terms of *operations* between radix-based large integers, it is important to recall that two messages can interact if they are encoded and encrypted with the same parameters. The majority of the arithmetic operations can be computed by using a schoolbook approach (homomorphically mixing linear operations and PBS) and by keeping an eye on the degree of fullness in each block. When carries are full, they need to be propagated to next block: this is done by extracting the carry and the message and adding carry to the next block (last carry can be thrown away). We provide some examples to better understand in Supplementary Material B.

When it comes to computing a generic LUT over a radix-based modular integer, the only known approach from the literature is the Tree-PBS from [GBA21], which becomes more and more complex when the number of blocks grows.

2.3.2 CRT-based large integers

The CRT based approach consists in encrypting a large integer modulo $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$ as a list of κ LWE ciphertexts, such that each pair β_i and $\beta_{j\neq i}$ of bases are co-primes. Each of the κ ciphertexts is defined according to a pair $\{\beta_i, p_i\}_{0\leq i<\kappa}$ such that $2 \leq \beta_i < p_i < q$.

In order to *encode* a message $\mathsf{msg} \in \mathbb{Z}_\Omega$, one needs to compute $\{m_i\}_{i=0}^{\kappa-1}$ such that $\mathsf{msg} = m_i \mod \beta_i$ for all $0 \leq i < \kappa$. Then we can independently encode and encrypt each m_i into an LWE ciphertext. To *decode*, we simply need to compute the modular reduction in base β_i and compute the inverse of the CRT.

With this CRT encoding, we have to empty the carry buffers when they are (almost) full. Indeed, the quantity overlapping the base β_i is not needed to maintain correctness but when using TFHE PBS, the bit of padding needs to be preserved.

We need to only call the message extraction algorithm, described in Section 2.2 when needed.

All the arithmetic *operations* can be performed independently on the blocks by using the operators described in Section 2.2. Concerning the evaluation of LUT, the only known way in the literature to compute them on CRT-based large integers, is the technique proposed by [KS21], that can be used only when the LUT to evaluate is CRT friendly. By CRT friendly, we intend a LUT L that can be independently evaluated in each component, i.e., L such that $\mathsf{Encode}_{\mathsf{CRT}}\left(L\left(\mathsf{msg}\right)\right) = \left(L_0\left(m_0\right), \cdots, L_{\kappa-1}\left(m_{\kappa-1}\right)\right)$ where $\mathsf{Encode}_{\mathsf{CRT}} = \left(m_0, \cdots, m_{\kappa-1}\right)$. For generic LUT evaluations, once again, the only technique known in the literature is the Tree-PBS by [GBA21].

Native CRT. In TFHE, we can also encode CRT integers by using no padding bit and no carry buffer (so no degree of fullness either), and by encoding the message m_i as $\left\lfloor \frac{q}{\beta_i} \cdot m_i \right\rceil$. By doing so, additions and scalar multiplications become native and do not require any PBS, except for noise reduction. To compute additions one can use the LWE addition on each residue, and to compute a scalar multiplication by α, one can decompose α with the CRT basis into smaller integers, and compute scalar multiplications with them. Without the bit of padding, the PBS can be evaluated only with a WoP-PBS algorithm. However, to evaluate generic LUT, the problem is still open. We will provide a solution in the next sections.

2.3.3 Limitations

The radix and CRT approaches discussed in this section are a first step towards solving the precision problem in TFHE-like schemes. However, they come with limitations:

- The radix approach is limited to the modulo Ω that can be expressed as a product of bases. But if the modulo is as instance a large prime, no solution is known.
- The CRT approach suffers from the CRT requirements, i.e., co-prime bases, and the precision limitation we have in practice with TFHE. Indeed, there are a limited number of primes between 2 and 128. It means that this approach is good when Ω is composed of small enough co-prime factors but for the rest of the possible Ω we need other solutions.
- For both the radix and the CRT approach, the only way to evaluate a generic LUT is the Tree-PBS, which does not scale well with the number of blocks, and so it is still inefficient in practice.

In Section 4 we provide solutions to overcome all these limitations.

3 Parameter Selection for FHE

To compute over ciphertexts one needs to select parameters related to their shape. With LWE ciphertexts a dimension n is required, and with GLWE ciphertexts, a polynomial size N and a dimension k are required. In this paper, those parameters are called *macro-parameters*. In addition, some FHE operators (definition 4) come with some degrees of freedom that one also needs to set. As an example, a key switch requires a base β and a level ℓ. Those parameters are called *micro-parameters*, because they are only used locally, inside an FHE operator.

Micro and macro parameters have an impact on the cost and/or the noise added during the evaluation of an FHE operator, so they need to be carefully picked. In a computation circuit, one needs to find parameters not only for one FHE operator, but for a graph of FHE operators. The more degrees of freedom in a DAG there are, the harder the parameter search is. The question we answer in this section is then:

For a given graph of FHE operators, how to find parameters so the evaluation is the fastest while preserving both correctness and security?

The task we are considering here is not trivial and was an open problem until this contribution.

Our framework takes as input a graph of FHE operators, a level of security and a correctness probability, and outputs parameters that will guarantee:

1. the desired *level of security*,
2. the *correctness* of the computation up to the desired correctness probability,
3. a *cost* as small as possible.

The first guarantee is easy to reach using the security oracle (definition 6) that can be built using the lattice-estimator [APS15]. Indeed, one can always increase the amount of noise at encryption (or key generation) to get the desired security. Using this, one does not need to find the best encryption noise, one can simply look for the best LWE dimension (or GLWE dimension and polynomial size) and take the minimal encryption noise given by the security oracle. In the end, one is sure to provide enough security as the noise is chosen with respect to other ciphertext parameters.

To guarantee the correctness of a computation (guarantee 2), one needs to rely on the noise model of each FHE operator in the graph. With FHE schemes, there is a link between the noise inside a ciphertext and the correctness of the computation. In fact, if the noise grows too much, the message will be tampered and the decryption algorithm will not yield the correct result. In order to guarantee the correctness, one needs to track the noise at each step of the computation (using the noise model) and choose parameters in a way that the noise remains small

enough.

The last guarantee is to have a cost as small as possible. For that, one needs to use the cost model and select the parameters that minimize this cost (among the ones that satisfy guarantee 2). Naturally, the more realistic the cost model is, the better the parameters will be in practice.

Basis for FHE Optimisation

In this paper, we will require a few higher level definitions. For instance, we formalize what an FHE operator is.

Definition 4 (FHE & Plain operator) *Any FHE operator $\mathbb{O}$ is an implementation of an FHE algorithm, on a given piece of hardware, taking as input some ciphertexts and/or plaintexts and returning one or more ciphertexts. A plain operator is a function mapping several integers into an output list of integers.*

Definition 5 (Noise & Cost Model) *FHE operators are associated with a noise model, a cost model and an plain operator. A noise model is often a formula used to model the noise evolution across an FHE operator. The cost model is a surrogate for the metric one wants to minimize, it could be the execution time, the power consumption, or the price. A cost is written* $\mathsf{Cost}(\cdot)$*. The FHE operator must compute the same operation as its associated plain operator under some noise constraints.*

Noise formulae and cost model. A noise formula for a given homomorphic operator takes as input the variance of the input ciphertext noises, some cryptographic parameters involved in the operator computation, as well as the plaintext values used in the operator.

The *noise of a freshly encrypted ciphertext* is a random (small) integer drawn from a given distribution $\chi(\sigma)$, where σ^2 is its variance. Variances help us quantifying noise in ciphertext, so whenever it is written that a ciphertext contains more noise than another, we mean that the noise inside the first ciphertext is drawn from a normal distribution with a bigger variance than the second one.

In this paper we will always consider the *cost model* to approximate the running time on a single thread. More details on the cost model used in the experiments / benchmarks are provided later in the paper. Other and more complex cost models could be considered (e.g. combining complexity of operations with keys and ciphertext sizes, pieces of hardware, RAM, etc.), but we leave this as a future work.

Security. The *security* of a GLWE-based scheme depends on the distribution of the secret key (for example binary, ternary or Gaussian), the product between the GLWE dimension and the polynomial size (i.e. $k \cdot N$), the noise distribution, and the ciphertext modulus (often written q). To estimate the security level offered by some given parameters one can use the LWE/Lattice-estimator [APS15]. As a general

rule of thumb, to keep the same security level, when increasing the product $k \cdot N$ we can decrease the minimal noise needed inside a ciphertext.

In the rest of this paper, we assume that, for each possible distributions of the secret key, we have access to the following security oracle:

Definition 6 (Security Oracle) *Given the product $k \cdot N$, a level of security λ and a ciphertext modulus q, the security oracle outputs the minimal noise variance σ^2_{min} needed in a ciphertext for it to be secure with the required level of security.*

3.1 The FHE Optimization Problem

We start by explaining the core ideas to ensure the three aforementioned guarantees.

As said above, one needs to choose the *macro-parameters* among a set of possible values. For example, the polynomial size N must be a power of 2. One wants to narrow it down to a finite set, and a practical yet wide enough space for TFHE-like schemes could be $\mathcal{P}_N = \{2^8, 2^9, \cdots, 2^{17}\}$. In the same manner, the LWE dimension n could be selected in $\mathcal{P}_n = [\![256, 2048]\!]$ and the GLWE dimension in $\mathcal{P}_k = [\![1, 6]\!]$. The $\mathcal{P}_N$ is called the *search space* of N.

Definition 7 (FHE DAG) *Let $\mathcal{G} = (V, L)$ be a DAG of FHE operators. We define $V = \{\mathcal{O}_i\}_{1 \leq i \leq \alpha}$ as the set of vertices, each of them being an FHE operator. We define L as the set of edges, each of them associated with the modulus p of the encrypted message i.e. $L \subset \{\{x, y, p\} \,|(x, y) \in V^2, p \in \mathbb{N}\}$. When L is not needed, we will simply write $\mathcal{G} = V$. We note $\mathsf{Cost}\,(\mathcal{G}, x)$ the cost of running the FHE graph $\mathcal{G}$ with the parameter set x.*

For a given FHE DAG $\mathcal{G}$ (definition 7), one also needs to set the *micro parameters*. For example, the logarithm of the decomposition base for a KS or a PBS $\log_2(\beta)$ can be taken in $\mathcal{P}_{\log_2(\beta)} = [\![1, \lfloor \log_2(q) \rfloor]\!]$ and the level of the decomposition ℓ in $\mathcal{P}_\ell = [\![1, \lfloor \log_2(q) \rfloor]\!]$. As (β, ℓ) are used to do a radix decomposition of each integer composing the input ciphertext, we know that $\ell \cdot \log_2(\beta) \leq \log_2(q)$ so in practice, we will consider (β, ℓ) as one unique variable in $\mathcal{P}_{\log_2(\beta),\ell} = \{(\log_2(\beta), \ell) \in [\![1, \lfloor \log_2(q) \rfloor]\!]^2,\ \ell \cdot \log_2(\beta) \leq \log_2(q)\}$.

In the end, one needs to choose a set of parameters in the Cartesian product of the search spaces of all the micro and macro parameters of a graph $\mathcal{G}$. This space is noted $\mathcal{P}_\mathcal{G}$ and is called the *search space* of $\mathcal{G}$. In the rest of the paper, this set is simply called $\mathcal{P}$ when there is no ambiguity on the graph.

Definition 8 (Noise Bound) *Let $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\widetilde{M}\right)$ a GLWE ciphertext of an encoding $\widetilde{M}$ of M with a message modulus p and π padding bits. The noise bound $t_\alpha(\pi, p)$ for a failure probability α is the biggest integer satisfying:*

$$\sigma \leq t_\alpha(\pi, p) \Rightarrow \mathbb{P}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right) \leq \alpha$$

The noise bound can also depend on other values, for instance it could take the degree of fullness later defined in this paper.

Remark 2 *Assuming that the input ciphertext contains a noise polynomial* $E = \sum_{i=0}^{N-1} e_i X^i \in \mathfrak{R}_q$ *such that* $\forall i, e_i \sim \mathcal{N}(0, \sigma^2)$*, we have an explicit formula for the noise bound* $t_\alpha(\pi, p) = \frac{\Delta}{2 \cdot \kappa}$ *with* $\kappa = z^*(p_{\mathsf{fail}})$*, the standard score (Definition 1) for* $p_{\mathsf{fail}} = 1 - \sqrt[N]{1-\alpha}$*. Let us assume* $\sigma \leq t_\alpha$*. Immediately using Definition 1 and Equation 1, we have* $\mathbb{P}\left(|e_i| \geq \frac{\Delta}{2}\right) \leq p_{\mathsf{fail}} = 1 - \sqrt[N]{1-\alpha}$*. Thus*

$$\begin{aligned}
\mathbb{P}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right) &= \mathbb{P}\left(\bigcup |e_i| \geq \frac{\Delta}{2}\right) \\
&= 1 - \mathbb{P}\left(\bigcap |e_i| < \frac{\Delta}{2}\right) \\
&= 1 - \prod_{i=1}^{N} \mathbb{P}\left(|e_i| < \frac{\Delta}{2}\right) \text{ by indep. of } \{e_i\}_{i \in [\![1,N]\!]} \\
&= 1 - \prod_{i=1}^{N} \left(1 - \mathbb{P}\left(|e_i| \geq \frac{\Delta}{2}\right)\right) \\
&\leq 1 - (1 - p_{\mathsf{fail}})^{\frac{1}{N}} = \alpha
\end{aligned}$$

When the input ciphertext is an LWE ciphertext i.e. $N = 1$*, we have* $1 - \sqrt[N]{1-\alpha} = \alpha$*.*

Using the noise bound, we can guarantee a correct decoding up to a given probability using only the distribution of the noise which can be publicly estimated. The tightness of the noise model is crucial to build tight confidence intervals.

Every ciphertext in an FHE DAG must have a noise smaller than its associated noise bound in order to guarantee the correctness of the computation. With those constraints, we define the *noise feasible set*, a subset of the search space $\mathcal{P}$ where every set of parameters will guarantee a correct computation.

Definition 9 (Noise Feasible Set) *Let* $\mathcal{G}$*, an FHE DAG such that* $\mathcal{G} = (V, L)$ *with* $L = \{(\cdot, \cdot, p_i)\}_{i \in [\![1, |L|]\!]}$*, and let* α *be a failure probability. Let* $\{\sigma_i\}_{i \in |L|}$ *be the standard deviation of the noise in the ciphertexts transiting on every edge of* $\mathcal{G}$*. For every edge* i*, we must have* $\sigma_i(x) \leq t_\alpha(p_i)$ *which defines a subset of the search space* $\mathcal{P}$*:* $\mathcal{S}_i = \{x \in \mathcal{P} | \sigma_i(x) \leq t_\alpha(p_i)\}$*. The intersection of all those sets is the noise feasible set* $\mathcal{S}$*: the set of parameter sets that will lead to a correct computation. We have:*

$$\mathcal{S} = \bigcap_{i \in I} \mathcal{S}_i = \{x \in \mathcal{P} | \forall i \in [\![1, |L|]\!], \sigma_i \leq t_\alpha(p_i)\}$$

By choosing a set of parameters that is in the noise feasible set, we are sure to have a correct computation which satisfies guarantee 2. In this set, we want to find the set of parameters minimizing the cost of the FHE DAG. Formally, we want:

$$\arg\min_{x \in \mathcal{P}} \mathsf{Cost}(\mathcal{G}, x) \text{ s.t. } x \in \mathcal{S}(\mathcal{G}) \tag{2}$$

The problem of finding efficient and correct FHE parameters is then a minimization problem under constraints. We can naturally use optimization techniques to solve it. The issue is that the complexity of the problem is dependent on the size of the FHE DAG which can rapidly become unrealistic for large DAGs. In the next section, we present several non trivial simplifications prior to the optimization enabling to speed up the task.

Remark 3 *As we defined a feasible set for the noise, we can also define other feasible sets for other constraints. For instance to limit the size of the public keys (key switching keys, bootstrapping keys, ...), the size of the ciphertexts (bandwidth) or even to add some constraints between parameters.*

3.2 Pre-Optimization & Graph Transformations

To simplify the optimization problem, we present an analysis working on any FHE DAG. The idea is to subdivide it in sub-graphs with the constraint that to compute the noise distribution of a ciphertext in one of these sub-graphs, we do not need to know the noise distribution of a ciphertext in another sub-graphs. The starting point is to note that there are some FHE operators that output ciphertexts with a noise independent of the input noise for some well-chosen parameters. This motivates us to distinguish those FHE operators from the rest:

Definition 10 (FHE operator Categories) *We divide the FHE operators (definition 4) into two categories regarding their respective noise formulae:*

(i) an operator which outputs a noise independent of the input noise, such as the PBS in our context;

(ii) an operator which adds some noise to the input noise, such as a KS or a dot product;

Using this distinction, for any FHE DAG, we can identify sub-graphs that are independent from others. Now that we have several independent sub-graphs, we want to find a way to compare them together. To do so, we define the notion of *atomic pattern types* to regroup sub-graphs of FHE operators called *atomic patterns* that we know how to compare. For instance, two atomic patterns of the same type can have a different message modulus p or different number of inputs.

For each atomic pattern types, we will compare atomic patterns and identify the ones where the noise will be the highest. Those are the ones we need to take into account when trying to construct $\mathcal{S}(\mathcal{G})$.

Definition 11 (Atomic Pattern Type) *An Atomic Pattern (AP) type $\mathcal{A}^{(\cdot)}$ corresponds to a sub-graph of FHE operators that outputs one or several ciphertexts with a noise independent of the input noise.*

An Atomic Pattern A is a particular instance of an AP type $\mathcal{A}^{(\cdot)}$. When an AP $A \in \mathcal{A}^{(\cdot)}$ is instantiated with a parameter set x, we write $A(x)$. From $A(x)$ one can estimate the amount of noise at any edge of its FHE sub-graph and one can also estimate its total cost using a cost model.

Once we have identified the atomic pattern types in a graph $\mathfrak{G} = (V, E)$, we can build a FHE DAG $\mathfrak{G}' = (V', E')$ such that each FHE operator in V' is an atomic pattern i.e. $V' = \{A_i(\cdot)\}_{i \in [\![1, |V'|]\!]}$. This new graph is equivalent to the input graph and we have $\mathcal{S}(\mathfrak{G}) = \bigcap_{i \in [\![1, |V'|]\!]} \mathcal{S}(A_i(\cdot))$. We leverage the fact that we can compare the noise between atomic patterns of the same type to efficiently find the atomic patterns that have the smallest feasible sets. We will describe this procedure for a noise feasible set, but this can be extended to another kind of feasible set - for instance, the evaluation key sizes.

Two AP of the same type can be compared even without a given set of parameters. Hence we can introduce the notion of *domination between AP.*

Definition 12 (AP Domination) *An AP A dominates A' if any $x \in \mathcal{P}(\mathfrak{G})$ satisfying the noise constraints of A also satisfies the constraints of A'. More formally, we have $\mathcal{S}(A) \subset \mathcal{S}(A')$ i.e. $\mathcal{S}(A) \bigcap \mathcal{S}(A') = \mathcal{S}(A)$. A' is said to be dominated by A*

For all AP types in a graph $\mathfrak{G}$, for all APs of this type, we can simply keep the ones that are not dominated by any other AP. Indeed, we can discard the APs that are dominated because their constraints will be satisfied if the constraints of one of their dominant AP are satisfied.

With TFHE, we mainly use three FHE operators: the homomorphic dot product (DP), the key switch and the programmable bootstrapping. The key switch is generally computed before the PBS (as in [CJP21]). We consider the noise formulae of [CLOT21] for the key switch and the bootstrapping. Because of the FFT in TFHE PBS, we had to add a corrective formula to take into account the noise added by the floating point representation. In particular, simply by casting the bootstrapping key from a *64-bit integers* to a *float* (represented with 64 bits) some of the LSB are lost. Similarly, the error grows all along computations in the Fourier domain due to the floating point arithmetic. To correct the formula accordingly, one solution is to collect data regarding the noise in many different parameter settings and use them to deduce a corrective formula that takes into account the FFT-induced error. Using this method, we found that the following formula provides a good correction for the variance of the output of a bootstrapping: $n \cdot 2^{\omega_1} \cdot \ell \cdot \beta^2 \cdot N^2 \cdot (k+1)$ with $\omega_1 \approx 22 - 2.6$ (where 22 is $2 \cdot (64 - 53)$, since $q = 2^{64}$, 53 corresponds to the mantissa bits in the f64 floating point representation, and 2.6 is an experimental fitting).

All the experiments and benchmarks later provided in this paper will consider the algorithmic complexity of each FHE algorithm for the cost model. It means that we count the number of additions, multiplications, castings between integer types, and the asymptotic cost of the FFT in each algorithm and use it as a surrogate of the execution time. For instance, the operation $\sum_{i=1}^{\alpha} M_i \cdot \mathsf{CT}_i$ (where $M_i \in \mathfrak{R}_q$ are polynomials and $\mathsf{CT}_i = (A_i, B_i) \in \mathfrak{R}_q^2$ are RLWE ciphertexts,) will have a cost of

$$\underbrace{(2+1)\ \alpha\ N \log(N)}_{\text{to FFT domain}} + \underbrace{2\alpha\ N}_{\text{float} \times} + \underbrace{(\alpha - 1)N}_{\text{float} +} + \underbrace{2\ N \log(N)}_{\text{to standard domain}}$$

With this cost model, we assume the cost of a multiplication between floating point numbers or integers to be same than the cost of an addition between integers. While

this hypothesis is false in practice, it is close enough to provide efficient parameter sets. To simplify the problem, we assume the cost of the dot product to be negligible compared to the other FHE operators. Here, we assume the cost of an atomic pattern A to be the sum of the cost of every FHE operator inside it, i.e. the cost of a PBS and the cost of a KS.

A homomorphic dot product is a dot product between a vector of ciphertexts and a vector of integers. Notice that given some ciphertexts $\{\mathsf{ct}_i\}_{i\in I}$ with independent noises coming from $\mathcal{N}(0,\sigma^2)$ and some weights $\{\omega_i\}_{i\in I}$, the noise in the output ciphertext $\mathsf{ct}_{\mathsf{out}} = \sum_{i\in I} \mathsf{ct}_i \cdot \omega_i$ follows the distribution $\mathcal{N}(0,\nu^2\sigma^2)$ with $\nu_2^2 = \sum_{i\in I}\omega_i^2$, the squared 2-norm. Thus, given a dot product between a vector of ciphertexts with the same (normal) noise distribution and a vector of integers, we only need the 2-norm ν to characterize the output noise of a dot product.

Naturally, we define our first concrete atomic pattern type $\mathscr{A}^{(\mathrm{CJP21})}$ which is composed of a DP, followed by a KS and a final PBS (i.e. a MS, a BR and a SE) as in [CJP21]. Here we assume every input of the dot product to be the output of a bootstrapping, hence we do not consider the fact that some of those inputs could be freshly-encrypted ciphertext. Everything we describe below is easily modifiable to take that into account. In the definition of the dot product, we saw that the 2-norm ν and the input variance are sufficient to compute the output noise of a DP if every input ciphertext has the same normal noise distribution. Hence, an atomic pattern AP of type $\mathscr{A}^{(\mathrm{CJP21})}$ is entirely characterized by two values: the 2-norm ν and its noise bound t. We will note $A = A(\nu, t)$.

It is easy to compare the noise in atomic patterns of this type using the following property which is a special case of definition 12.

Theorem 1 (AP Domination) *Let's consider $A_1, A_2 \in \mathscr{A}^{(\cdot)}$ two AP of a type that include a homomorphic DP, ν_1, ν_2 two 2-norms such that $\nu_1 \leq \nu_2$ and t_1, t_2 two noise bounds where $t_2 \leq t_1$. We have: $\mathcal{S}(A_2(\nu_2, t_2)) \subset \mathcal{S}(A_1(\nu_1, t_1))$ i.e. $\mathcal{S}(A_2(\nu_2, t_2)) \bigcap \mathcal{S}(A_1(\nu_1, t_1)) = \mathcal{S}(A_2(\nu_2, t_2))$. A_1 is said to be dominated by A_2*

Proof 2 (Sketch) *A_1 and A_2 share the same type. When decreasing the noise bound, i.e. going from t_1 to t_2, we have less possible solutions x, but all the ones that satisfy t_2 will satisfy t_1. The same reasoning works for the 2-norms. By increasing the 2-norm, i.e. going from ν_1 to ν_2, there are less possible solutions x, but all the solutions satisfying ν_2 will satisfy ν_1.* □

Given a graph $\mathcal{G} = \{A_i\}_{i\in I}$ of atomic patterns of type $\mathscr{A}^{(\mathrm{CJP21})}$, we can apply the theorem above to simplify the construction of $\mathcal{S}(\mathcal{G})$. In fact, we do not need to build each $\mathcal{S}(A_i)$ as some of them are included in others. From our input graph $\mathcal{G}$, we construct a new graph $\mathcal{G}_{\mathsf{pareto}} = \mathsf{Pareto}(\mathcal{G}) = \{A'_i\}_{i\in I_{\mathsf{pareto}}}$ containing only non-dominated atomic patterns using theorem 1 (Pareto comes from *Pareto front*, well known in optimization). It follows that $\mathcal{G}_{\mathsf{pareto}}$ contains at most as many atomic patterns as there are different noise bounds in the graph.

An interesting property of $\mathcal{G}_{\mathsf{pareto}}$ is that $\mathcal{S}(\mathcal{G}) = \mathcal{S}(\mathcal{G}_{\mathsf{pareto}})$ i.e. if one solves the optimization problem (Eq. 2) using $\mathcal{S}(\mathcal{G}_{\mathsf{pareto}})$ instead of $\mathcal{S}(\mathcal{G})$, we will get the

same optimal solution. This is interesting because to compute $\mathcal{S}(\mathcal{G}) = \bigcap_{i \in I} \mathcal{S}(A_i)$ we needed to build $|I|$ search spaces and with $\mathcal{G}_{\mathsf{pareto}}$, we only need to build $|I_{\mathsf{pareto}}|$ search spaces and most of the time $|I| \gg |I_{\mathsf{pareto}}|$.

Another useful observation is to notice that in an atomic pattern of type $\mathcal{A}^{(\mathrm{CJP21})}$, the noise is strictly increasing until the end of the modulus switching step in the final PBS. As the noise bound is assumed to be constant inside one atomic pattern, we do not need to check that the noise satisfies the noise bound t after the dot product or after the key switching, we only need to do it after the modulus switch. If we note $\sigma_{\mathsf{MS},1}$, the standard deviation of the noise after the modulus switching in an atomic pattern A_1, we have $\mathcal{S}(A_1) = \{x \in \mathcal{P} | \sigma_{\mathsf{MS},1}(x) \leq t\}$.

As we assume the cost of a dot product to be negligible, the cost of an atomic pattern is only dependent on the cryptographic set of parameters and not on a particular instance of an atomic pattern of type $\mathcal{A}^{(\mathrm{CJP21})}$.

For a graph $\mathcal{G} = \{A_i\}_{i \in I}$, we have $\mathsf{Cost}(\mathcal{G}, x) = \sum_{i \in I} \mathsf{Cost}(A_i, x)$ for x a solution in the search space $\mathcal{P}$ and we now that for any $(i, j) \in I^2$, $\mathsf{Cost}(A_i, x) = \mathsf{Cost}(A_j, x)$, so instead of minimizing the cost of running the total graph $\mathcal{G}$, we can settle for minimizing the cost of one atomic pattern of type $\mathcal{A}^{(\mathrm{CJP21})}$.

To sum up, for a given graph $\mathcal{G}$, instead of solving equation 2, we can build a new graph $\mathcal{G}_{\mathsf{pareto}}$ as described above and solve the following which will give us the same value but will be easier to compute.

$$\arg\min_{x \in \mathcal{P}} \mathsf{Cost}(\cdot, x) \text{ s.t. } x \in \mathcal{S}(\mathcal{G}_{\mathsf{pareto}}) \tag{3}$$

The above problem is greatly simplified but still depends on the input graph $\mathcal{G} = \{A(\nu_i, t_i)\}_{i \in I}$. It can be useful to have access to sets of parameters that work for a wide range of applications. Given a graph $\mathcal{G}$, we will be able to select the best set of parameters in those pre-computed sets.

A simple way to do that is to introduce another special graph $\mathcal{G}_{\mathsf{worst}}$, that we call the *worst case* atomic pattern. It is defined as $\mathcal{G}_{\mathsf{worst}} = \{A(\max_{i \in I} \nu_i, \min_{i \in I} t_i)\}$. This graph is reduced to only one atomic pattern that may or may not be present if the input graph $\mathcal{G}$. Using theorem 1, we know that $\mathcal{S}(\mathcal{G}_{\mathsf{worst}}) \subset \mathcal{S}(\mathcal{G})$. So if we solve equation 3 on $\mathcal{G}_{\mathsf{worst}}$, we end up with a feasible solution for $\mathcal{G}$. Using this new graph, we are able to pre-compute sets of cryptographic parameters for different values of (ν, t). Given a graph $\mathcal{G}$, we will select the set of parameters for the worst case atomic pattern $\mathcal{G}_{\mathsf{worst}}$ of $\mathcal{G}$.

Above, we found a feasible solution and intuitively, this solution is close to the optimal one. To have bounds on the optimality of the solution for a graph $\mathcal{G} = \{A_i\}_{i \in I}$, we can use another particular graph $\mathcal{G}_{\mathsf{best}}$ defined as $\mathcal{G}_{\mathsf{best}} = \{A(\nu^*, t^*)\}$ with $t^* = \min_{i \in I} t_i$ and $\nu^* = \max\{\nu_i | A(\nu_i, t^*) \subset \mathcal{G}\}$ i.e. it is a graph composed of the atomic pattern of the graph $\mathcal{G}$ that have the smallest noise bound and the highest norm2 for this noise bound. If the worst case atomic pattern is the same as the best case atomic pattern, the method described above yields an optimal solution as $\mathcal{G}_{\mathsf{pareto}} = \mathcal{G}_{\mathsf{worst}} = \mathcal{G}_{\mathsf{best}}$. If they are different, we can deduce a bound of

optimality: as $\mathcal{G}_{\mathsf{best}} \subset \mathcal{G}$, we know that $\mathcal{S}(\mathcal{G}) \subset \mathcal{S}(\mathcal{G}_{\mathsf{best}})$. Solving equation 3 for $\mathcal{G}_{\mathsf{best}}$ give us a lower bound on the cost of the optimal solution of equation 3 for $\mathcal{G}$ and solving equation 3 for $\mathcal{G}_{\mathsf{worst}}$ give us an upper bound.

The atomic pattern types give us a powerful tool to compare several variants of the bootstrapping existing in the FHE literature. As different bootstrapping techniques have different cost-noise trade-offs, it is hard to compare them. By studying atomic patterns, we do not need to trouble ourselves with that, if one bootstrap yields more noise than another, it will be taken into account as the input noise of the atomic pattern will be higher.

3.3 Takeaways On Larger Precision

We defined another atomic pattern type $\mathcal{A}^{(\mathrm{GBA21})}$ composed of a dot product, a key switch and the tree-PBS introduced in [GBA21]. The only way to compare the PBS of [CGGI20] in $\mathcal{A}^{(\mathrm{CJP21})}$ and the tree-PBS in $\mathcal{A}^{(\mathrm{GBA21})}$ is by solving equation 3 for the two types of atomic patterns with a range of 2-norms and a range of message precision, and finally plot the results.

In the Figure 5, we display the comparison between $\mathcal{A}^{(\mathrm{CJP21})}$ and $\mathcal{A}^{(\mathrm{GBA21})}$ for 4 distinct 2-norms and for message precision in $\{2^1, \cdots, 2^{24}\}$. The padding bit is not included in the message precision.

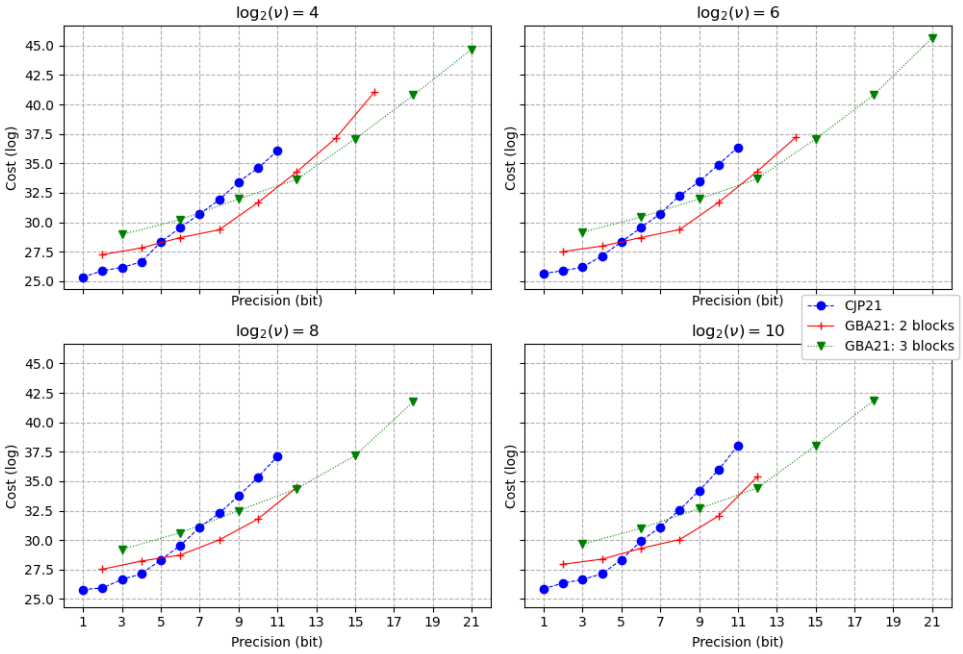

Figure 5: In this figure, we compare the cost of AP type $\mathcal{A}^{(\mathrm{CJP21})}$ and AP of type $\mathcal{A}^{(\mathrm{GBA21})}$ with 2 and 3 blocks.

In this experiment, we choose $\mathcal{P}(N) = \{2^1, \cdots, 2^{18}\}$, the search space of the polynomial size N. We set $q = 2^{64}$ and we used a probability of failure $p_{\mathsf{fail}} \approx 2^{-35}$ and one bit of padding (i.e. $\pi = 1$).

Remark 4 (Noise Bound) *For $\mathcal{A}^{(CJP21)}$, the noise bound (definition 8) is defined as $t(p,1) = \frac{q}{2^{1+1} \cdot p \cdot z^*(p_{\mathsf{fail}})}$.*

For $\mathcal{A}^{(GBA21)}$, the noise bound needs to be computed differently because this AP with 2 blocks (respectively 3 blocks) involves η_2 (respectively η_3) PBS, all sources of potential failures.

$$\eta_i = i \cdot \frac{p^{i-1} - 1}{p - 1} + 1, \text{ with } i \text{ the number of blocks}$$

To guarantee a global failure probability for one $\mathcal{A}^{(GBA21)}$, the noise bound needs to be computed from the number η_i of PBS. We start by computing the failure probability needed for one PBS defined as $p'_i = 1-(1-p_{\mathsf{fail}})^{\frac{1}{\eta_i}}$ and from it we can finally compute the noise bound for each PBS $t(p,1) = \frac{q}{2^{1+1} \cdot p \cdot z^*\left(p'_i\right)}$. A generalization of this approach is explained in Section 5.2.

The first takeaway is that TFHE bootstrapping (in atomic pattern $\mathcal{A}^{(\mathrm{CJP21})}$, blue/• curve) can only handle messages up to 11 bits of precision. By using these parameters set, the cost of this atomic pattern with regards to the precision is an exponential function in two parts. For precisions above 4 to 5 bits (padding bit not included), adding a bit of precision more than doubles the cost, indeed the polynomial size doubles for every additional bit of precision. TFHE PBS does not scale well with the precision, to maximize efficiency, it should not be used when the messages have more than 5 bits of precision.

For $\mathcal{A}^{(\mathrm{GBA21})}$, we used on the first layer the multi-value PBS introduced in [CIM19] and we used PBS over encrypted lookup tables [CGGI20] on the other layers. The tree-PBS of [GBA21] takes as input a vector of ciphertexts each containing part of the message. The red/+ curve (respectively green/▼ curve) represents the cost to compute a tree-PBS over 2 ciphertexts (respectively 3 ciphertexts) each one containing a chunk of the message. Using this, we can reach precisions that are not feasible with the bootstrapping from [CGGI20]. Above 11 bits, we cannot find parameters that will guarantee the correctness of $\mathcal{A}^{(\mathrm{CJP21})}$. Regarding the tree-PBS with 2 blocks, it becomes interesting in term of cost with 6 bits of precision or more, and offers parameters up to 16 bits of precision. For higher precision, no feasible solution could be found. The tree-PBS with 3 blocks provides a way to go above that and we found solutions for precision up to 21 bits. It is more efficient than the other two starting at 10 bits of precision. It is important to notice that even if solutions exist, computing $\mathcal{A}^{(\mathrm{GBA21})}$ over message of 21 bits costs more than 2^{20} times the cost of [CGGI20] PBS over Boolean messages.

To conclude this comparison, [CGGI20]'s bootstrapping used as in [CJP21] (i.e. with a KS before and not after) is the best way to apply a function over message of small precision (1 to 5 bits). For precision above 11 bits, we have to use the tree-PBS in [GBA21]. But as we can see in the figures, we need an algorithm more efficient than [GBA21] when it becomes too expensive, i.e., above 9 bits, especially if one wants to build efficient operations over larger homomorphic integers with TFHE and still being able to compute LUTs on them.

4 TFHE-based Large Integers

One of the main observations from the Section 3 concern the precision of messages that we can encrypt in TFHE-like ciphertexts. Thanks to the optimizer, we observed that it is more efficient to split a message into several ciphertexts instead of making the parameters of a single ciphertext grow (starting from 6 bits of precision).

The literature proposes already some solutions based on the radix decomposition and on the CRT decomposition, that we describe in Section 2. However, these solutions present some limitations. Indeed, the radix approach can be only used with a modulo that can be expressed as a product of small integers, and no generic solution is proposed for, as instance, large prime moduli. The precision limitations to 8-bits in the message space makes also the CRT approach very limited, since the number of prime or co-prime numbers smaller than 8 bits is really small. Additionally, when it comes to the generic LUT evaluation in these two contexts, the only known technique is the Tree-PBS [GBA21], which unfortunately does not scale well when the number of ciphertexts encrypting a message increases.

In order to overcome all these limitations, in this section we start by *generalizing the radix approach* to every modular integer, by proposing homomorphic modular reductions for radix-based ciphertexts. We then try to get the best of both worlds from the new radix approach and from the CRT by proposing a *new hybrid method*, which has no more limitations in terms of precision. More importantly, we also present a *new WoP-PBS* technique, i.e., a PBS without bit of padding, that we can use to evaluate generic LUTs on these new large integers and that performs better that the Tree-PBS for larger precisions. We provide details on the comparison of the two techniques and we conclude this section by presenting some *benchmarks*.

4.1 Generalization of large integer representations

In order to overcome the limitations in radix and CRT approaches for large integers in TFHE, we propose two improvements. We start from a generalization of the radix approach to any large modulus Ω. Then, we propose a hybrid approach that takes the best of both the radix and the CRT approaches and allows us to work with any moduli efficiently. In practice without the first improvement, the number of possible CRT residues is majorized by the numbers of small prime integers, thus harshly restricting the available general modulo Ω offered by the hybrid approach.

4.1.1 Generalization of radix to any large modulus Ω

By using the radix representation, homomorphic modular integers are defined modulus Ω, that is equal to the product of the bases $\beta_i \in \mathbb{N}, i \in [0, \kappa-1]$, i.e., $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$. Here, we propose to remove this restriction by generalizing the previous arithmetic to any modulus Ω s.t. $\prod_{j=0}^{\kappa-2} \beta_j < \Omega < \prod_{j=0}^{\kappa-1} \beta_j$. The only difference with the previous approaches lies in the computation of the modular reduction. In what follows,

we propose two complementary methods to perform this modular reduction, whose efficiency depends on Ω and the product of the selected basis.

First method for modular reduction. The first method consists in performing multiple LUT evaluations in the most significant block to reduce it modulo Ω. Indeed, the modular reduction is applied on the κ^{th} block (i.e., $\mathsf{ct}_{\kappa-1}$) which represents $m_{\kappa-1} \cdot \prod_{i=0}^{\kappa-2} \beta_i$ with $m_{\kappa-1} < p_{\kappa-1}$, and which might be larger than Ω. The complete process is detailed in Algorithm 1. The modular reduction is performed as a series of κ PBS (with KS, Line 2) and the result is a radix-based integer with a base $(\beta_0, \ldots, \beta_{\kappa-1})$ decomposition. The final step is to add the first $\kappa - 1$ blocks of the result of the modular reduction to the first $\kappa - 1$ blocks of the input (Line 4) and to replace the last block in the result by the $(\kappa - 1)$-th block obtained in the modular reduction (Line 5).

Algorithm 1: $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}) \leftarrow \mathsf{ModReduction}_1((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB})$

Context: $\begin{cases} P_j: \text{ r-redundant LUT for } \begin{cases} \mathbb{Z}_{p_{\kappa-1}} \to \mathbb{Z}_{\beta_j} \\ x \mapsto x'_j = \mathsf{Decomp}_j\left(x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega\right) \end{cases} \\ x'_j \text{ is the j-th element in the decomposition in base } (\beta_0, \ldots, \beta_{\kappa-1}) \\ \text{s.t. } x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = x'_0 + \sum_{i=1}^{\kappa-1} x'_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right) \end{cases}$

Input: $\begin{cases} (\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \text{ encrypting } \mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_i\right) \\ \text{s.t. } \mathsf{ct}_i \text{ encrypts message } m_i \text{ with parameters } (\beta_i, p_i) \\ \mathsf{PUB}\text{: public material for KS-PBS} \end{cases}$

Output: $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1})$, encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_i\right) \mod \Omega$

```
   /* Decompose message in block κ − 1 with respect to base (β_0, ..., β_{κ−1})   */
1  for j ∈ ⟦0; κ − 1⟧ do
2      c_j ← KS-PBS(ct_{κ−1}, PUB, P_j)
   /* Add (as in Section 2.2) decomposition to all the blocks up to κ − 2          */
3  for j ∈ ⟦0; κ − 2⟧ do
4      ct'_j ← Add(ct_j, c_j)
   /* Replace κ − 1 block with κ − 1 element in decomposition                      */
5  ct'_{κ−1} ← c_{κ−1}
6  return (ct'_0, ..., ct'_{κ−1})
```

Observe that the κ KS-PBS in Line 2 of Algorithm 1 could be replaced by optimized procedures evaluating several different LUT on the same input ciphertext. A few constructions have been proposed in the literature, such as the PBSmanyLUT [CLOT21] or the multi-value bootstrapping [CIM19].

Proof 3 (Correctness of Algorithm 1) *By construction, we have that* $\prod_{j=0}^{\kappa-2} \beta_j < \Omega < \prod_{j=0}^{\kappa-1} \beta_j$. *Then, reducing the* $(\kappa - 1)$*-th block encrypting the message* $m_{\kappa-1} < p_{\kappa-1}$, *rescaled by the product* $\prod_{i=0}^{\kappa-2} \beta_i$ *modulus* Ω *is enough to correctly clear its carry space without loosing information. This is homomorphically*

done by evaluating the κ *functions* $x \in \mathbb{Z}_{p_{\kappa-1}} \mapsto \mathsf{Decomp}_j\left(x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega\right)$ *with* $j \in [\![0, \kappa-1]\!]$. *Then, for all* $i \in [\![0; \kappa-1]\!]$, $\mathsf{Decrypt}(c_i) = r_i$, *giving* $r = r_0 + \sum_{i=1}^{\kappa-1} r_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right)$ *with* $r_i < \beta_i$ *and* $0 \leq r < \Omega$. *The last step is to compute the addition between each* ct_i *but the* $(\kappa-1)$*-th with the* c_i. *The final output is given by* $\mathsf{ct}' = (\mathsf{ct}'_0, \ldots \mathsf{ct}'_{\kappa-1})$. *Then, for all* $i \in [\![0; \kappa-2]\!]$, $\mathsf{Decrypt}(\mathsf{ct}'_i) = m_i + r_i$, *such that* $\mathsf{Decrypt}(\mathsf{ct}') = m_0 + r_0 + \sum_{i=1}^{\kappa-2}(m_i + r_i) \cdot \left(\prod_{j=0}^{i-1} \beta_j\right) + r_{k-1} \prod_{j=0}^{\kappa-1} \beta_j$. □

Second method for modular reduction. The second method idea is based on the shape of $-\prod_{h=0}^{\kappa-2} \beta_h$ (i.e., the negation of the scaling factor of the message in the $\kappa - 1$ block) reduced modulo Ω. The radix decomposition is:

$$\prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = \nu_0 + \nu_1 \cdot \beta_0 + \nu_2 \cdot \beta_0\beta_1 + \ldots + \nu_{\kappa-1} \cdot \prod_{j=0}^{\kappa-2} \beta_j.$$

If $\nu_{\kappa-1} = 0$ and the other elements of the decomposition, i.e., $\nu_0, \nu_1, \ldots, \nu_{\kappa-2}$, are small integers (ideally many of them set to 0), then this method is more efficient. Indeed, when these conditions are respected, the idea is to replace the MSB block by multiplying it by the non-zero constants ν_j and subtracting the results from the j-th input block, for $j \in [\![0, \kappa-2]\!]$. Some multiplications with positive constants are needed and might require some carry propagation prior to them depending on the degrees of fullness. This method is detailed in Algorithm 2. In the general case where the bases for each block are different, the algorithm performs a homomorphic decomposition into the right base, corresponding to a series of PBSs, that is detailed in Supplementary Material C. This step could be skipped if the bases are compatible. The padding algorithm that follows is simply a padding with zero ciphertexts for the addition and subtraction to work.
Let's develop the algorithm for a 3-blocks integer:

$$m = m_0 + m_1\beta_0 + m_2\beta_0\beta_1 \text{ and } \beta_0\beta_1 = \nu_0 + \nu_1\beta_0 + \nu_2\beta_0\beta_1 \bmod \Omega$$

therefore,

$$\begin{aligned} m &= m_0 + m_1\beta_0 + m_2\nu_0 + m_2\nu_1\beta_0 + m_2\nu_2\beta_0\beta_1 \\ &= (m_0 + m_2\nu_0) + (m_1 + m_2\nu_1)\beta_0 + (m_2\nu_2)\beta_0\beta_1 \mod \Omega. \end{aligned}$$

What happens is that if $\nu_2 = 0$, then we will have emptied the last block. Let's try this method on an example: $\Omega = 1055$, $\kappa = 3$, $\vec{\beta} = (\beta, \beta, \beta)$ with $\beta = 2^5$ and $\vec{p} = (p, p, p)$ with $p = 2^7$. Observe that $(2^5)^2 \mod 1055 = -31 = -1 \cdot 2^5 + 1$ (so $\nu_0 = 1$, $\nu_1 = -1$ and $\nu_2 = 0$). Then, the new reduced ciphertext would be composed by:

- in the block 0: the addition between the previous block 0 and the previous block 2 multiplied times 1;
- in the block 1: the addition between the previous block 1 and the previous block 2 multiplied times -1;

- in the block 2: an encryption of 0.

Observe that in Algorithm 2 the subtraction algorithm follows the regular school-book subtraction modulo an integer Ω.

Proof 4 (Correctness of Algorithm 2) *If the degree of fullness of the input $(\kappa-1)$-th block is small enough to be able to perform a constant multiplication times the largest of the constants $\nu_0, \ldots, \nu_{\kappa-2}$, followed by a homomorphic addition, the algorithm can start. In fact, the algorithm consists in multiplying the non-zero constants ν_j times the block $\mathsf{ct}_{\kappa-1}$ and then to subtract the result to the input ct_j block, for $j \in [\![0, \kappa-2]\!]$. The result of this operation, by definition of the constants $\nu_0, \ldots, \nu_{\kappa-1}$, is a new radix-based encryption of msg reduced modulo Ω. In case the bases in the blocks are not the same, a homomorphic decomposition step (as described in Supplementary Material C) needs to be performed before addition.*

As for Algorithm 1, this new ciphertext is not a "fresh" ciphertext, in the sense that the carries in the blocks are not all empty (because of the homomorphic addition). A carry propagation step can be applied if necessary and it can be used to continue the computations. □

4.1.2 Larger Integer using Hybrid Representation

As we explained above, the CRT-only approach has some limitations. To overcome them, we create a new homomorphic hybrid representation that mixes the CRT-based approach with the radix-based approach, in order to take advantage of the best of both worlds. The idea is to use the CRT approach as the top layer in the structure, and to represent the CRT residues by using radix-based modular integers when needed: with this approach we do not have any more restrictions on Ω.

Encode. Let $(\Omega_0, \cdots, \Omega_{\kappa-1})$ be integers co-primes to each other, i.e., (Ω_i, Ω_j) co-primes for all $i \neq j$, and let $\Omega = \prod_{i=0}^{\kappa-1} \Omega_i$. To encode a message $\mathsf{msg} \in \mathbb{Z}_\Omega$, as in the CRT-only approach, the message is split into a list of $\{\mathsf{msg}_i\}_{i=0}^{\kappa-1}$ such that $\mathsf{msg}_i = \mathsf{msg} \mod \Omega_i$ for all $0 \leq i < \kappa$. At this point, for each message msg_i for $i \in [\![0, \kappa-1]\!]$, the encoding used for radix-based modular integers is used (Encode from definition 2). Then, any CRT residues Ω_i have its own list of radix bases: $(\beta_{i,\kappa_i-1}, \cdots, \beta_{i,0})$ and more generally its parameters $\{(\beta_{i,j}, p_{i,j})\}_{0 \leq j < \kappa_i} \in \mathbb{N}^{2\kappa_i}$. The formal encoding is described in the Figure 6.

Decode. The decoding is done in two steps: first, each independent radix-based modular integer is decoded to obtain the independent residues modulo $\Omega_0, \ldots, \Omega_{\kappa-1}$, and then the CRT is inverted to retrieve the message modulo Ω.

Arithmetic operations. To perform any homomorphic operation, it is enough to perform the computation on each radix component independently, as shown for the CRT-only approach. Then, depending on the Ω_i values, the modular reduction algorithms (i.e., Alg. 1 or Alg. 2) can be used.

$$\mathsf{msg} \mod \Omega \mapsto \begin{cases} \mathsf{msg}_0 = \mathsf{msg} \mod \Omega_0 \mapsto \begin{cases} \{m_{0,j}\}_{j=0}^{\kappa_0-1} \text{ s.t.} \\ \mathsf{msg}_0 = m_{0,0} + \sum_{j=1}^{\kappa_0-1} m_{0,j} \cdot \left(\prod_{k=0}^{j-1} \beta_{0,k}\right) \\ \text{and } \tilde{m}_{0,j} = \mathsf{Encode}\left(m_{0,j}, p_{0,j}, q\right) \\ \forall 0 \leq j < \kappa_0 \end{cases} \\ \vdots \\ \mathsf{msg}_{\kappa-1} = \mathsf{msg} \mod \Omega_{\kappa-1} \mapsto \begin{cases} \{m_{\kappa-1,j}\}_{j=0}^{\kappa_{\kappa-1}-1} \text{ s.t.} \\ \mathsf{msg}_{\kappa-1} = m_{\kappa-1,0} + \sum_{j=1}^{\kappa_{\kappa-1}-1} m_{0,j} \cdot \left(\prod_{k=0}^{j-1} \beta_{\kappa-1,k}\right) \\ \text{and } \tilde{m}_{\kappa-1,j} = \mathsf{Encode}\left(m_{\kappa-1,j}, p_{\kappa-1,j}, q\right) \\ \forall 0 \leq j < \kappa_{\kappa-1} \end{cases} \end{cases}$$

Figure 6: Hybrid approach visualisation combining CRT representation on the top level and radix representation below.

The hybrid approach can be seen as a generalization of both the CRT-only approach (if $\kappa_i = 1$ for all $0 \leq i < \kappa$) and the pure radix-based modular integer approach (if $\kappa = 1$). It also covers the mixed cases where some of the κ_i are equal to 1 and the others are greater.

For generic LUT evaluation, the only known solution is the Tree-PBS [GBA21]. In next section we propose a new technique for generic LUT evaluation and we show that it scales better than the Tree-PBS.

4.2 LUT evaluation over large integers

The PBS [CGGI20, CJP21] takes as input a single LWE ciphertext and output the LUT evaluation on the encrypted message. However, when the message is encoded in multiple LWE ciphertexts, a single PBS is not enough. The Tree-PBS method proposed in 2021 by Guimarães, Borin and Aranha [GBA21] enables to evaluate a large look-up table over many input ciphertexts. For completeness, we provide details about how to use this technique for our large homomorphic integers in Supplementary Material C.1. The Tree-PBS is a valid solution for the evaluation of generic LUT in large integers, but its complexity increases exponentially with the number of blocks (i.e., the number of LWE ciphertext composing a large integer ciphertext). Additionally, the Tree-PBS technique uses the classical PBS [CGGI20, CJP21], which has the constraints on the bit of padding and on the small precision of the messages.

In this section, we propose an alternative technique to evaluate generic LUTs on large integers, that scales better than the Tree-PBS and does not have the constraint on the bit of padding.

4.2.1 New WoP-PBS

A WoP-PBS, i.e., a PBS which does not require a bit of padding, is a method that was introduced for the first time by Chillotti et al. [CLOT21]. As the classical PBS, it takes as input an LWE ciphertext with or without bits of padding in the MSB,

a public key called bootstrapping key, and a LUT L. It outputs the homomorphic evaluation of the LUT on the input message, i.e., an LWE encryption of $L[m]$.

Here, we propose a new WoP-PBS that is able to take as input not only one LWE ciphertext but several, it is able to round (or truncate or more) each of the input messages to a given precision, and it can be used to compute several LUT on the same set of inputs at the cost of (about) a single LUT.

Our method is based on two building blocks: the circuit bootstrapping and the mixed (or vertical or horizontal) packing from [CGGI20]. In practice, the algorithm executes the following steps:

- It starts by using generalized PBS [CLOT21], evaluating a scaled sign function (negacyclic), and homomorphic subtraction to extract all the bits of the encrypted large message. Each bit is output as a LWE ciphertext.
- It converts each of the LWE ciphertexts extracted by previous step into GGSW ciphertexts, by using circuit bootstrapping [CGGI20].
- It uses the GGSW ciphertexts from previous step to evaluate the LUT as a mixed (or vertical or horizontal) packing [CGGI20]: it consists in practice in a CMux tree, followed by a blind rotation and one (or several) sample extraction.

The cleartext representation of the new WoP-PBS is presented in Figure 7.

In general, the circuit bootstrapping is the most expensive part of the algorithm (each circuit bootstrapping requires several PBSs, each followed by several functional key switchings). Since the number of circuit bootstrapping corresponds to the number of bits composing the input message, the technique generally scales linearly in the size of the input message. However, after a certain input size, the mixed packing stops being negligible and becomes as costly (or even more) than the circuit bootstrapping part: roughly speaking, this happens when the number of CMuxes in the mixed packing part becomes as big as the number of CMuxes in the PBSs computed inside the circuit bootstrappings (e.g., for the parameter sets that we use in our experiments, this happens when the input size is about 28 bits).

We provide the details of the technique (using vertical packing in this case) in Algorithm 3. To evaluate several LUT, we just need to repeat the vertical packing for each LUT evaluation.

Proof 5 (Correctness of Algorithm 3) *In what follows, we prove that the output of Algorithm 3 is:* $\mathsf{ct}_{\mathsf{out}} = (\mathsf{LWE}_s(l_0(m)), \cdots, \mathsf{LWE}_s(l_{\kappa-1}(m)))$*, with* $l_i \in \mathbb{Z}_\omega$ *a LUT.*

The first step is called the Bit Extract (corresponding to the lines 3 to 9 in algorithm 3): all bit of information from all the κ^{th} *bits are going to be extracted to be stored in a new block. The first extracted bit is the least significant of the message. To do so, it is first shifted to the MSB of the ciphertext. More formally, the first step is to shift the* α_i*-th MSB to the* 1*-st MSB by multiplying* ct_i *by* $2^{\alpha_i - 1}$ *with* $\alpha_i = \lceil \log_2(\beta_i) \rceil$*. At this point, the next step would be to compute a PBS with a*

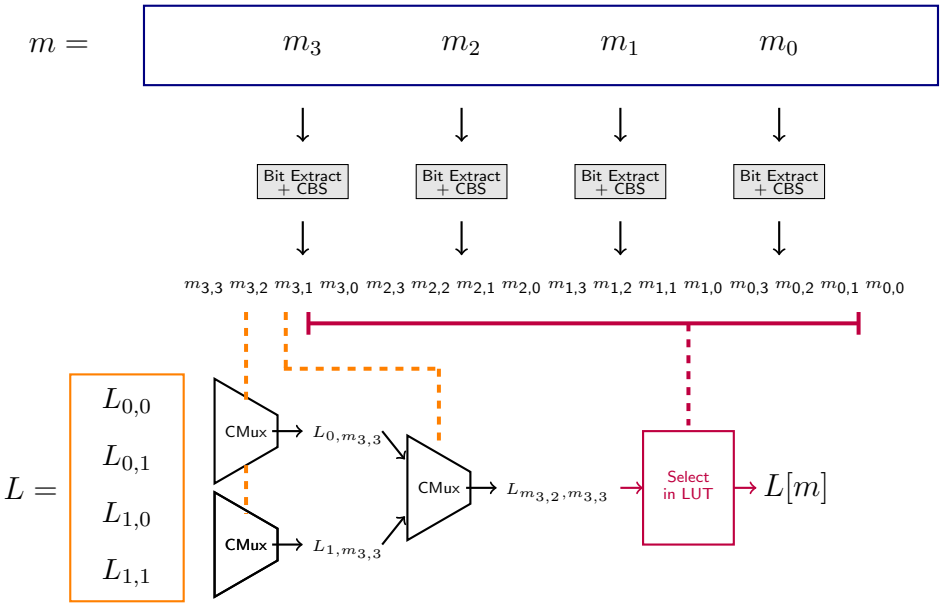

Figure 7: Cleartext evaluation of the new WoP-PBS (toy example). The values $m_{i,j}$ (for $i,j \in \{0,1,2,3\}$) are bits. We split the LUT L into 4 smaller LUTs $(L_{0,0}, L_{0,1}, L_{1,0}, L_{1,1})$ to be evaluated in the CMux tree. The output LUT of this tree is given in input to the operation selecting the right output of a LUT (corresponding to the blind rotation). The output $L[m]$ is the element in the LUT L corresponding to the input message m. The Bit Extract blocks correspond to the lines 2 to 12 in Algorithm 3 and the CMux tree followed by a blind rotation corresponds to the vertical packing (VPLut, line 13).

LUT defined by the polynomial $P(X) = -\frac{q}{2^{\alpha_i+1}} \cdot \sum_{i=0}^{N-1} X^i$ *s.t.:*

$$\begin{cases} \frac{q}{2^{\alpha+1}} & \text{if } \mathsf{Decrypt}(\mathsf{ct}_i \cdot 2^{\alpha_i-1}) \in [\frac{q}{2}, q[\\ \frac{-q}{2^{\alpha+1}} & \text{otherwise} \end{cases}$$

Then, by homomorphically adding $\frac{q}{2^{\alpha+1}}$*, this gives either an encryption of* $\frac{q}{2^{\alpha}}$ *or* 0*. However, observe that for a cleartext equals to* 0*, any negative noise will lead to a wrong result in the extracted bit (i.e., after decryption, we get* $\frac{q}{2^{\alpha+1}}$ *instead of* $-\frac{q}{2^{\alpha+1}}$*). To avoid this type of errors we need to add a small correction integer denoted* ϵ_i *(for* $i \in [0, \kappa-1]$*) to the shifted ciphertext* $\mathsf{ct}_i \cdot 2^{\alpha-1}$*. In what follows, we compute the value of the corrective term.*

We need to be cautious about encoded values that are closed to the two bounds 0 *or* $\frac{q}{2}$*: if the noise is negative, then the PBS will return an incorrect result. In order to choose the right correcting term, we need to determine the smaller distance (denoted* $\mathsf{d}(\cdot,\cdot)$*) between the preceding encoded value* v_1 *and* q*, and the preceding encoded value* v_2 *and* $\frac{q}{2}$*, i.e.:*

$$\min\left(\mathsf{d}\left(v_1 \in \left[0; \frac{q}{2}\right[, \frac{q}{2}\right), \mathsf{d}\left(v_2 \in \left[\frac{q}{2}; q\right[, q\right)\right)$$

.

We now compute the two distances. At this point, we need to distinct between two cases:

1. β_i ***is a power of two***, *i.e.,* $\beta_i = 2^k$. *The values are encoded by* $\left\lfloor \frac{q}{2^k} \cdot i \right\rceil \mod q$ *with* $i \in [\![0, 2^k[\![$. *For the shift we compute* $\left\lfloor \frac{q}{2^k} \cdot i \right\rceil \cdot 2^{k-1} \mod q$, *so the only remaining encoded values are* 0 *and* $\frac{q}{2}$, *so the distance between these two values is* $d = \frac{q}{2} = \left\lfloor \frac{q \cdot 2^{k-1}}{\beta_i} \right\rfloor = 2\epsilon_i$.

2. β_i ***is not a power of two***, *i.e.,* $\beta_i = \beta_i' \cdot 2^k$, *with some odd* $\beta_i' \neq 1$. *As* $\beta_i << q$, *after the first shift, for all* $j \in \mathbb{N}$ *we obtain the following bound over the encoded values:*

$$\begin{aligned}\left\lfloor \frac{q}{\beta_i} \cdot (j \mod \beta_i) \right\rfloor \cdot 2^{\alpha_i - 1} \mod q &\leq \left(\frac{q}{\beta_i} \cdot (j \mod \beta_i) + \frac{1}{2} \right) \cdot 2^{\alpha_i - 1} \mod q \\ &\leq \frac{q}{\beta_i} \cdot (j \mod \beta_i) \cdot 2^{\alpha_i - 1} + 2^{\alpha_i - 2} \mod q\end{aligned}$$

We now want to work with β_i' *instead of* β_i. *Then, for all* $j \in \mathbb{N}, \exists j', j'' \in \mathbb{N}$, *such that:*

$$\begin{aligned}\frac{q}{\beta_i} \cdot (j \mod \beta_i) \cdot 2^{\alpha_i - 1} \mod q &= \frac{q}{\beta_i'} \cdot (j' \mod \beta_i') \cdot 2^{\alpha_i' - 1} \mod q. \\ &= \frac{q}{\beta_i'} \cdot (j'' \mod \beta_i') \mod q.\end{aligned}$$

The next is step is to compute the minimum of the distances:

$$\min \left(\mathsf{d}\left(v_1 \in [0; \frac{q}{2}[, \frac{q}{2} \right), \mathsf{d}\left(v_2 \in \left[\frac{q}{2}; q \right[, q \right) \right) - 2^{\alpha_i - 2}, v_i \in \left\{ \frac{q}{\beta_i'} \cdot j' \mod q \right\}_{j' \in \{0, \beta_i' - 1\}}$$

.

First we can bound v_1*:*

$$v_1 \leq \left\lceil \frac{q}{\beta_i'} \cdot \left\lfloor \frac{\beta_i'}{2} \right\rfloor \right\rceil = \frac{q}{2} - \left\lfloor \frac{q}{2\beta_i'} \right\rfloor$$

So we have $d_1 = \mathsf{d}\left(v_1 \in [0; \frac{q}{2}[, \frac{q}{2} \right) \geq \left\lfloor \frac{q}{2\beta_i'} \right\rfloor$.

Next we can bound v_2*:*

$$v_2 \leq \left\lceil \frac{q}{\beta_i'} \cdot (\beta_i' - 1) \right\rceil = q - \left\lfloor \frac{q}{\beta_i'} \right\rfloor$$

So we have $d_2 = \mathsf{d}\left(v_2 \in [\frac{q}{2}; q[, q \right) > \left\lfloor \frac{q}{\beta_i'} \right\rfloor \geq \left\lfloor \frac{q}{2\beta_i'} \right\rfloor$. *The distance is then bounded by* $\left\lfloor \frac{q}{2\beta_i'} \right\rfloor - 2^{\alpha_i - 2}$. *The correcting term is finally defined as half of this bound, i.e.,* $\epsilon_i = \left\lfloor \frac{q}{4\beta_i'} \right\rfloor - 2^{\alpha_i - 3} = \left\lfloor \frac{q \cdot 2^{k-2}}{\beta_i} \right\rfloor - 2^{\alpha_i - 3}$.

Remark *Since the term 2^{α_i-3} is very small regarding q, it can be neglected to have the same ϵ in the both cases. About the noise bound, this term is also negligible, since it is smaller than 1 before the shift.*

By taking $\epsilon_i = \left\lfloor \frac{q \cdot 2^{k-2}}{\beta_i} \right\rfloor$ and adding ϵ_i to $\mathsf{ct}_i \cdot 2^{\alpha-1}$ we ensure that for any message, an error e of size $|e| < \epsilon_i$ will lead to a correct PBS evaluation. This means that before the shift, the noise in ct_i should be smaller than $\left\lfloor \frac{q \cdot 2^{k-2}}{\beta_i} \right\rfloor \cdot 2^{-\lfloor \log_2(\beta_i) \rfloor}$.

At this point, the less significant bit (the α-th bit) has been extracted and stored into a new $\mathsf{LWE}_{i,\alpha}$. To extract the next bit, we first subtract $\mathsf{LWE}_{i,\alpha}$ to ct_i. With this operation we ensure that the α-th bit is now equal to 0. As we want to extract the $(\alpha-1)$-th bit, we now shift by $2^{\alpha-2}$. Finding the corrective term ϵ_i is much easier in this case, as the second bit is equal to 0 after the shift. Hence, we can take $\epsilon = \frac{q}{4}$ and extract the bit with a PBS. To extract the remaining bits, we just need to repeat the previous steps (subtraction, shift, add $\epsilon = \frac{q}{4}$ and PBS).

Concerning the correctness of circuit bootstrap and vertical packing, we refer to [CGGI20].

□

The noise of the output of Algorithm 3 corresponds to the noise of a circuit bootstrapping – a PBS, followed by a private functional KS (i.e., an external product) – followed by $\sum_{i=0}^{\kappa-1} \delta_i$ CMuxes (all the keys are uniformly binary). The formula can be obtained from the noise formulae presented in [CLOT21] and it is equal to:

$$\begin{aligned}\mathsf{Var}(E_{\mathbf{CB}}) = &\underbrace{n\ell_{\mathbf{BR}}(k+1)N\frac{\beta_{\mathbf{BR}}^2+2}{12}\mathsf{Var}(\mathsf{BSK}) + n\frac{q^2-\beta_{\mathbf{BR}}^{2\ell_{\mathbf{BR}}}}{24\beta_{\mathbf{BR}}^{2\ell_{\mathbf{BR}}}}\left(1+\frac{kN}{2}\right)+}_{\mathbf{PBS}}\\ &\underbrace{+\frac{nkN}{32}+\frac{n}{16}\left(1-\frac{kN}{2}\right)^2}_{\mathbf{PBS}} + \underbrace{\ell_{\mathbf{BR}}(n+1)\frac{\beta_{\mathbf{BR}}^2+2}{12}\mathsf{Var}(\mathsf{KSK})+}_{\text{private functional }\mathbf{KS}}\\ &\underbrace{+\frac{q^2-\beta_{\mathbf{BR}}^{2\ell_{\mathbf{BR}}}}{24\beta_{\mathbf{BR}}^{2\ell_{\mathbf{BR}}}}\left(1+\frac{n}{2}\right)+\frac{n}{32}+\frac{1}{16}\left(1-\frac{n}{2}\right)^2}_{\text{private functional }\mathbf{KS}}\end{aligned}$$

$$\begin{aligned}\mathsf{Var}(E_{\mathsf{WoP\text{-}PBS}}) = \mathsf{Var}(E_{\mathbf{CB}}) &+ \underbrace{\left(\sum_{i=0}^{\kappa-1}\delta_i\right)\ell_{\mathbf{CB}}(k+1)N\frac{\beta_{\mathbf{CB}}^2+2}{12}\mathsf{Var}(E_{\mathbf{CB}}) + \left(\sum_{i=0}^{\kappa-1}\delta_i\right)\frac{kN}{32}+}_{\text{mixed packing}}\\ &\underbrace{+\left(\sum_{i=0}^{\kappa-1}\delta_i\right)\frac{q^2-\beta_{\mathbf{CB}}^{2\ell_{\mathbf{CB}}}}{24\beta_{\mathbf{CB}}^{2\ell_{\mathbf{CB}}}}\left(1+\frac{kN}{2}\right)+\frac{\left(\sum_{i=0}^{\kappa-1}\delta_i\right)}{16}\left(1-\frac{kN}{2}\right)^2}_{\text{mixed packing}}\end{aligned}$$

The cost of Algorithm 3 corresponds to the cost of $\sum_{i=0}^{\kappa-1}(\delta_i - 1)$ KS-PBSs (with parameters n, k, N, $\ell_{\mathbf{BR}}$, $\beta_{\mathbf{BR}}$, $\ell_{\mathbf{KS}}$, $\beta_{\mathbf{KS}}$, σ_{BSK}, σ_{KSK}), plus the cost of $\sum_{i=0}^{\kappa-1}\delta_i$ circuit bootstrappings (with parameters n, k, N, $\ell_{\mathbf{BR}}$, $\beta_{\mathbf{BR}}$, $\ell_{\mathbf{CB}}$, $\beta_{\mathbf{CB}}$, σ_{BSK}, σ_{FPKS}), plus the cost of $\log_2(N) + 2^{\sum_{i=0}^{\kappa-1}(\delta_i)-\log_2(N)} - 1$ CMuxes (with parameters k, N, $\ell_{\mathbf{CB}}$, $\beta_{\mathbf{CB}}$). Observe that the base and level used in the PBS for bit extraction and in

the PBS for circuit bootstrapping might be chosen differently. Several optimizations are possible in Algorithm 3. We did not include them directly in Algorithm 3 to simplify the explanation:

- The PBSs in the first step of the algorithm can either be computed independently, or sequentially, from LSB to MSB, by removing an extracted bit from the input ciphertext before extracting the next one.
- The second step of the circuit bootstrapping, which is a series of several packing functional key switchings, can be improved by following a similar footstep as a technique proposed in [CCR19]. We perform an initial LWE-to-GLWE KS (not functional) to each of the outputs of the PBS, and then, as already done in [CCR19], we perform an external product times the GGSW encryption of the GLWE secret key to obtain the remaining GLWE ciphertexts. This allows us to reduce the size of public evaluation keys at the cost of a slightly larger noise in the output.
- The KS-PBS performed in Line 8 is a Generalized PBS, as described in [CLOT21], so the modulus switching directly reads the next bit to be extracted. The sign function is evaluated in order to re-scale the bit at the right scaling factor. The circuit bootstrappings used in Lines 11 and 12 are also instantiated with a Generalized PBS. If we chose a value of $\vartheta > 0$ we could improve the circuit bootstrappings with a PBSmanyLUT, as described in [CLOT21], i.e., perform all the PBS in a circuit bootstrapping at the cost of a single PBS. Using this technique imposes an additional constraint on the noise in input of the circuit bootstrapping.
- We can observe that one of the PBS of the circuit bootstrappings used in Line 11 could be avoided thanks to the KS-PBS in Line 8, that might already provide the bit extracted at the right re-scaling factor.

Remark 5 *In general, the number of circuit bootstrappings performed in Algorithm 3 corresponds to the number of bits of the input message. However, this number might be slightly larger in some special cases, such as the case where the carry buffers have not been emptied beforehand, or the case of native CRT. In these cases, we might need to extract more bits of information, and so perform more PBSs during bit extraction and more circuit bootstrappings. Furthermore, different possible inputs might encode the same value, hence the LUT L needs to contain some kind of redundancy. If the goal is to compute the discrete function f, one needs to compute the L as $L[(m_0, \cdots, m_{\kappa-1})] = \mathsf{Encode}\left(f\left(\mathsf{Decode}\left(m_0, \cdots, m_{\kappa-1}\right)\right)\right)$.*

Remark 6 (Faster Algorithm 3 for Special LUTs) *Observe that the new WoP-PBS approach can be also adapted, and be very convenient, for particular LUTs such as the ReLU or the sign function in the radix mode, as instance. Indeed, for these functions we are only interested in the MSB part of the message, so the mixed packing is greatly simplified, and the cost of the WoP-PBS becomes linear in the number of blocks.*

4.2.2 Fast & Native CRT Implementation

Following up on what we present in Section 2.3.2, it is possible to have a fast version of CRT encoded integers spread out into several ciphertexts. As an example, to encode a residue $m_0 = 3 \mod 7$, its associated plaintext is $\lfloor 3\frac{q}{7} \rceil$. Additions and scalar multiplications are extremely fast in this context: they do not require any PBS and they can be computed independently and in parallel on each of the CRT residues with fast FHE operators. Indeed, to compute additions we use the LWE addition on each residue, and to compute a scalar multiplication by α, we decompose α with our CRT basis into smaller integers, and compute scalar multiplications with them.

To the best of our knowledge, until this work, there were no efficient algorithm to compute LUT in this context. However, one can use the new WoP-PBS to do so. They simply need to extract the first $\lceil \log_2(\beta_i) \rceil$ most significant bits for each residue, thanks to negacyclic sign functions. A bit extraction on a non-2-power encoded ciphertext has to be computed in the exact same manner.

The sign evaluation for an integer message m encoded as $\tilde{m} = m \cdot \frac{q}{p}$ (where both q and p are powers of two) is computed by adding to the input ciphertext $\frac{q}{2p}$, computing a PBS on it with a trivial encryption of $P(X) = \sum_{i=0}^{N-1} -\frac{q}{2p'} X^i$ and finally adding to the output $\frac{q}{2p'}$. The output is the encoding $b \cdot \frac{q}{p'}$ where b is the most significant bit of m. With p not a two power, one needs to replace $\frac{q}{2p}$ with $\left\lfloor \frac{q}{2p} \right\rfloor$.

We built modular integers with 16 bits of precision with the CRT basis $(\beta_0 = 7, \beta_1 = 8, \beta_2 = 9, \beta_3 = 11, \beta_4 = 13)$ and we used for each odd residue a non power-of-2 encoding.

4.2.3 Comparison Between $\mathcal{A}^{(\text{this work})}$, $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{GBA21})}$

In Section 4.2.1, we introduced a new WoP-PBS in Algorithm 3. We can now resume our comparison, started in Section 3.3, to find out which algorithm is the best (depending on some parameters) to compute over ciphertexts with large precision. To do so, we consider a new atomic pattern type $\mathcal{A}^{(\text{this work})}$ composed of a DP and the WoP-PBS (Algorithm 3). As this algorithm can work on a single ciphertext or on several ciphertexts containing chunks of the message, we present three variants: 1, 2 and 4 blocks. We display a comparison between $\mathcal{A}^{(\text{CJP21})}$, $\mathcal{A}^{(\text{GBA21})}$ and $\mathcal{A}^{(\text{this work})}$ on figure 8. We used the exact same context as in Figure 5 for this experiment, so the failure probability is for the three of them $p_{\mathsf{fail}} \approx 2^{-35}$.

Remark 7 (Noise Bound) *For $\mathcal{A}^{(CJP21)}$ and $\mathcal{A}^{(GBA21)}$, please refer to Remark 4. For $\mathcal{A}^{(this\ work)}$, we have a certain number of sequential bit extractions per input LWE ciphertext / block. In theory, we want to take into account all those potential PBS (one per bit extraction), but we noticed that the first one dominates all the others regarding noise. In fact, their impact on the total failure probability is negligible compared with the first bit extraction. Our experiments showed that for 2-norms $\nu \geq 4$, and for failure probability below 2^{-25} this assumption holds. We leave as future works the exploration of this topic. With this assumption, we start by computing the*

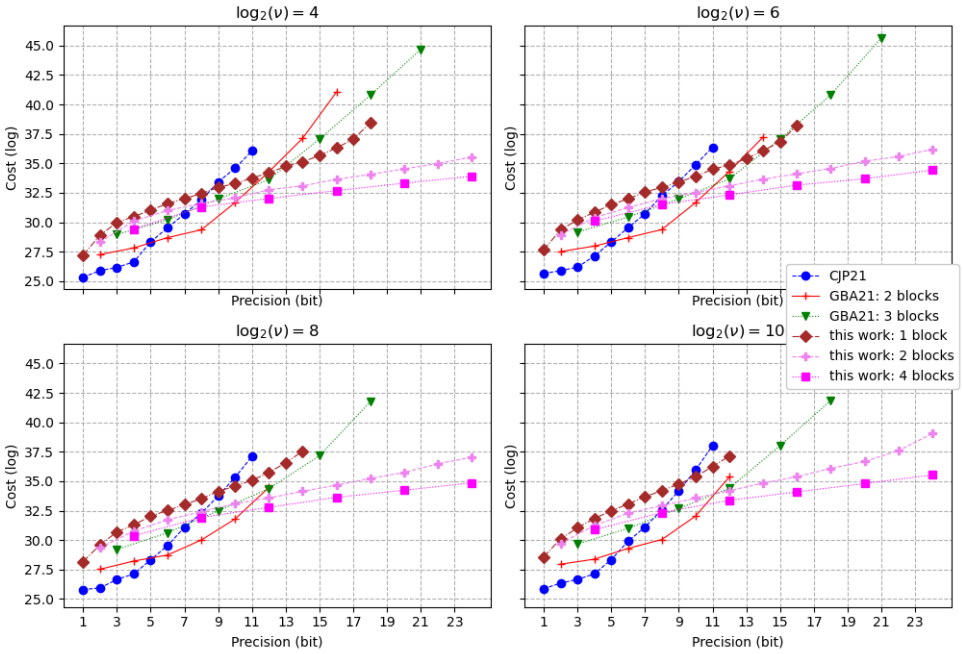

Figure 8: In this figure, we evaluate a LUT over a few encrypted inputs. We compare AP type $\mathcal{A}^{(\text{this work})}$, corresponding to the WoP-PBS introduced in this paper (1, 2 and 4 blocks), and AP type $\mathcal{A}^{(\text{GBA21})}$, corresponding to the Tree-PBS [GBA21] (2 and 3 blocks). As a baseline, AP of type $\mathcal{A}^{(\text{CJP21})}$ is also plotted.

failure probability needed for one PBS defined as $p'_i = 1-(1-p_{\mathsf{fail}})^{\frac{1}{\kappa}}$*, since there are* κ *input LWE ciphertexts. From it, we can finally compute the noise bound for each PBS* $t\,(p,0) = \frac{q}{2^1 \cdot p \cdot z^*\left(p'_i\right)}$.

The brown/♦ curve represents the cost of the best parameter set for an atomic pattern $\mathcal{A}^{(\text{this work})}$ working over one block. We can immediately notice that, between 1 and 9 bits of precision, $\mathcal{A}^{(\text{CJP21})}$ is more interesting than the new bootstrapping (Algorithm 3). However, with precisions from 10 bits and above, $\mathcal{A}^{(\text{this work})}$ has solutions that are more efficient than the $\mathcal{A}^{(\text{CJP21})}$'s existing ones, and finds solutions when $\mathcal{A}^{(\text{CJP21})}$ cannot. For small ν, it offers solutions that are slightly better than the ones from $\mathcal{A}^{(\text{GBA21})}$.

The pink/✚ curve (respectively the pink/■ curve) represents the atomic pattern $\mathcal{A}^{(\text{this work})}$ for two blocks (respectively four blocks) of message. On those curves, we see that it scales much better than the other atomic pattern types. With Algorithm 3, we manage to find solution up to 24 bits of precision. Those solutions are costly but far less than the ones for $\mathcal{A}^{(\text{GBA21})}$, and for comparison, it is only 2^{10} times more costly to compute a LUT over a message with 24-bits of precision with $\mathcal{A}^{(\text{this work})}$ than compute a LUT with 1-bit of precision with $\mathcal{A}^{(\text{CJP21})}$. Also for 18 bits of precision, the new WoP-PBS with two blocks is approximately 2^7 times faster than the tree-PBS in $\mathcal{A}^{(\text{GBA21})}$ with three blocks.

To sum up, for small precisions (up to 5 bits), TFHE PBS is the best option

among the three considered. Above 10/11 bits of precision, the algorithm we introduced in this paper (Algorithm 3) becomes the best alternative and improves the state of the art by a non-negligible factor.

Remark 8 (LUT Evaluation for Even More Precision) *It is important to observe that evaluating a LUT on integers larger than e.g., 30 bits, even in clear, becomes too expensive in terms of memory (e.g. a LUT for 30-bit input and output integers contains* $2^{30} \cdot 64$ *bits* $= 8$ *GB of information). So both techniques – Tree-PBS and our new WoP-PBS – are anyway not practical anymore.*

Remark 9 (Small Public Material for Algorithm 3) *An important observation to make about Algorithm 3 is that the size of the needed public material scales way better than a tree-PBS as in [GBA21]. As an example, for a total of 18 bits of precision we have a key of* 1.65 *GB for* $\mathcal{A}^{(GBA21)}$ *and a size of* 0.926 *GB for* $\mathcal{A}^{(this\ work)}$.

4.2.4 Comparison Between $\mathcal{A}^{(\text{this work})}$ and $\mathcal{A}^{(\text{LMP21})}$

A few WoP-PBS constructions have been proposed in the literature. Some works [KS21, LMP21] already compare them, but our optimization framework enables to truly do it by comparing them at the best of their efficiency. This can be done by putting each of them in a different atomic pattern type and finding optimal parameters for different 2-norms and precisions. To do so, we create one additional atomic pattern type called $\mathcal{A}^{(\text{LMP21})}$ composed of a DP, a KS and the WoP-PBS from [LMP21]. We used the exact same context as in Figure 5 for this experiment, so the failure probability is for the both of them $p_{\mathsf{fail}} \approx 2^{-35}$. We display in figure 9 the comparison between our new WoP-PBS (Algorithm 3, blue/• curve) in $\mathcal{A}^{(\text{this work})}$ and the WoP-PBS from [LMP21] in $\mathcal{A}^{(\text{LMP21})}$ (red/+ curve).

Remark 10 (Noise Bound) *For* $\mathcal{A}^{(this\ work)}$*, please refer to Remark 7. For* $\mathcal{A}^{(LMP21)}$*, we consider the two sequential PBS involved in the algorithms. They almost have the same amount of input noise and thus we assume that they both contribute equally to the overall failure probability. We experimented the two possible scenarios, (i) taking the first PBS's input noise for the computation or (ii) taking the second one. We did not observe any difference between the two approaches for the considered failure probabilities and 2-norms. We start by computing the failure probability needed for one PBS defined as* $p'_i = 1 - (1 - p_{\mathsf{fail}})^{\frac{1}{2}}$ *and from it we can finally compute the noise bound for each PBS* $t\,(p, 0) = \frac{q}{2^{1+1} \cdot p \cdot z^*\left(p'_i\right)}$.

The first thing that we learn on the WoP-PBS of [LMP21] is that it does not scale well with big precisions, which is not surprising as the algorithm uses as subroutine two PBS from [CGGI20] to compute the WoP-PBS. Thus, as for $\mathcal{A}^{(\text{CJP21})}$, for precisions above 10, we do not find any feasible solutions. We can also identify as before two parts on the curve, the first one for small precisions (1-8 bits) and a

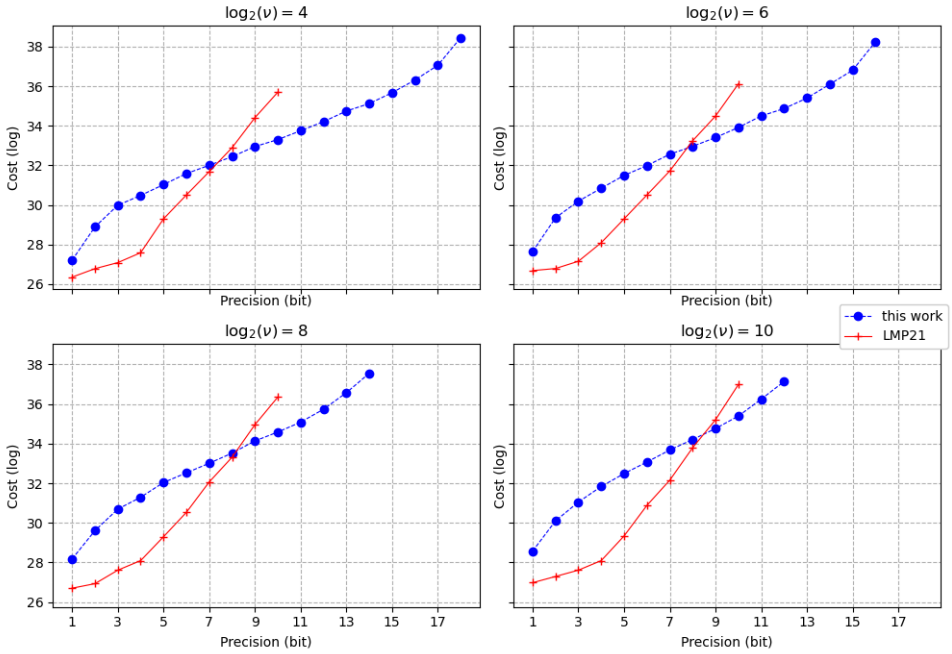

Figure 9: In this figure, we compare the cost of AP type $\mathcal{A}^{(\text{this work})}$ and type $\mathcal{A}^{(\text{LMP21})}$. The first one corresponds to DP-KS followed by our new WoP-PBS (Section 4.2.1), and the second one to DP-KS followed by the WoP-PBS from [LMP21].

second one for higher precisions: the reason behind this sudden growth in cost is also due to the increase of the polynomial size to manage bigger messages.

Thanks to the new WoP-PBS (Algorithm 3), we are able to compute a WoP-PBS over large messages. To conclude, this new algorithm scales better than existing algorithms to compute LUTs over large message and we do not need a padding bit which is a known constraint of TFHE bootstrapping.

4.3 Benchmarks

In this section, we provide a few practical benchmarks for integers of sizes 16 and 32 bits. All the cryptographic parameters are detailed in 4.3.1. The specifications of the machine are: Intel(R) Xeon(R) Platinum 8375C@2.90GHz with 504GB of RAM. Note that such an amount of RAM is not needed: all benchmarks can be run on a basic laptop. All implementations are done using TFHE-rs[4] (the follow-up of the Concrete library [CJL+20]).

4.3.1 Cryptographic Parameters

In Tables 1 and 2, we report the cryptographic parameters that we use to compute our benchmarks. All of them have been obtained with the optimization framework. In those tables, the notation $\mathfrak{B}$ (resp. ℓ) refers to the basis (resp. the number

[4] https://github.com/zama-ai/tfhe-rs

of levels) parameter used for a given FHE algorithm such as a key switch or a [CGGI20]'s PBS. By default, the cryptographic parameters ensure 128 bits of security, a failure probability $p_{\mathsf{fail}}\left(\mathcal{A}^{(\text{CJP21})}\right), p_{\mathsf{fail}}\left(\mathcal{A}^{(\text{this work})}\right) \leq 2^{-13.9}$ i.e. a standard score (Definition 1) of 4 which is pretty easy to experiment with.

Remark 11 (Biggest 2-Norm) *For a given message modulo β and carry-message modulo p one can find the worst 2-norm that they could encounter in the modular arithmetic defined in section 2.2. Indeed, a fresh encoding is at worst $\beta - 1$, and the biggest message one can consider before needing to empty the carry buffer is $p - 1$, so the biggest integer one can multiply a ciphertext with is $\left\lfloor \frac{p-1}{\beta-1} \right\rfloor$ which is the biggest 2-norm.*

param ID	AP parameters		LWE		GLWE			LWE-to-LWE key switch		PBS		WoP-PBS compatible
	p	ν	n	$\log_2(\sigma)$	k	$\log_2(N)$	$\log_2(\sigma)$	$\log_2(\mathfrak{B})$	ℓ	$\log_2(\mathfrak{B})$	ℓ	
# 1	2^2	3	615	−13.38	4	9	−51.49	2	5	12	3	#8
# 2	2^4	5	702	−15.69	2	10	−51.49	2	7	9	4	#9
# 3	2^6	5	872	−20.21	1	12	−62.00	4	4	22	1	#10
# 4	2^2	3	667	−14.76	6	8	−37.88	4	3	18	1	∅
# 5	2^4	5	784	−17.87	2	10	−51.49	4	3	23	1	∅
# 6	2^8	17	983	−23.17	1	14	−62.00	4	5	15	2	∅
# 7	2^6	9	838	−19.30	1	12	−62.00	3	5	15	2	∅

Table 1: Optimized parameters for AP of type $\mathcal{A}^{(\text{CJP21})}$.

param ID	AP parameters				LWE		GLWE			micro parameters		
	p	bit(s) to extract	κ	ν	n	$\log_2(\sigma)$	k	$\log_2(N)$	$\log_2(\sigma)$	operator	$\log_2(\mathfrak{B})$	ℓ
# 8 *compatible with CJP#1*	2^2	1	16	3	549	-11.62	2	10	-51.49	LWE-to-LWE key switch	2	5
										PBS	12	3
										packing key switch	17	2
										circuit bootstrapping	13	1
# 9 *compatible with CJP#2*	2^4	2	8	5	534	-11.22	2	10	-51.49	LWE-to-LWE key switch	2	5
										PBS	12	3
										packing key switch	17	2
										circuit bootstrapping	9	2
# 10 *compatible with CJP#3*	2^6	4	5	5	538	-11.33	4	10	-62.00	LWE-to-LWE key switch	1	10
										PBS	4	11
										packing key switch	20	2
										circuit bootstrapping	7	4
# 11	$\begin{pmatrix}7\\8\\9\\11\\13\end{pmatrix}$	$\begin{pmatrix}3\\3\\4\\4\\4\end{pmatrix}$	5	5	696	-15.53	2	10	-51.49	LWE-to-LWE key switch	2	7
										PBS	9	4
										packing key switch	17	2
										circuit bootstrapping	7	3
# 12	$\begin{pmatrix}3\\11\\13\\19\\23\\29\\31\\32\end{pmatrix}$	$\begin{pmatrix}2\\4\\4\\5\\5\\5\\5\\5\end{pmatrix}$	8	2^5	781	-17.79	1	11	-51.49	LWE-to-LWE key switch	1	16
										PBS	5	8
										packing key switch	13	3
										circuit bootstrapping	6	4

Table 2: Optimized parameters for AP of type $\mathcal{A}^{(\text{this work})}$.

In Table 1, we provide seven parameter sets for $\mathcal{A}^{(\text{CJP21})}$, each one with a bit of padding, a specific message modulus p and specific 2-norm ν. In Table 2, we provide five parameters sets for $\mathcal{A}^{(\text{this work})}$, each one with a specific (carry-)message modulo p, a specific number of bits to extract per LWE ciphertext during the WoP-PBS, a specific number κ of input LWE ciphertext to the WoP-PBS and a specific 2-norm ν. They do not have a bit of padding. In parameter IDs #11 and #12, the message modulus specifies the CRT base used and the corresponding number of bits to extract for each base.

Compatibility Between $\mathcal{A}^{(\text{this work})}$ and $\mathcal{A}^{(\text{CJP21})}$. We generated couples of parameter sets that are compatible, one for $\mathcal{A}^{(\text{CJP21})}$ and the other for $\mathcal{A}^{(\text{this work})}$. By compatible, we mean that one can go from one to the other freely and smoothly. From $\mathcal{A}^{(\text{CJP21})}$ to $\mathcal{A}^{(\text{this work})}$, one needs to remove the bit of padding in the usual LUT of $\mathcal{A}^{(\text{CJP21})}$'s PBS. From $\mathcal{A}^{(\text{this work})}$ to $\mathcal{A}^{(\text{CJP21})}$, one needs to add a bit of padding in the LUT of the usual $\mathcal{A}^{(\text{this work})}$'s WoP-PBS. But we also need other guarantees

to be able to freely compose atomic patterns $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{this work})}$. In particular, we need to guarantee that (i) each atomic pattern can absorb/deal with input noise either coming from $\mathcal{A}^{(\text{CJP21})}$ or $\mathcal{A}^{(\text{this work})}$ and (ii) the input LWE dimensions of each atomic pattern are compatible i.e. the product of the GLWE dimension k by the polynomial size N must be equal in both AP. We could remove constraint (ii) by adding two key switching keys, one to go from $\mathcal{A}^{(\text{CJP21})}$ to $\mathcal{A}^{(\text{this work})}$ and one to go from $\mathcal{A}^{(\text{this work})}$ to $\mathcal{A}^{(\text{CJP21})}$: we leave it as future work.

To satisfy those two conditions, we decided to first solve the optimization problem on $\mathcal{A}^{(\text{this work})}$ and later on $\mathcal{A}^{(\text{CJP21})}$ with more constraints. The first optimization gives us the product $k \cdot N$ and the output variance of $\mathcal{A}^{(\text{this work})}$. Then we solve the optimization problem for $\mathcal{A}^{(\text{CJP21})}$ with an additional constraint for the polynomial size N and the GLWE dimension k to satisfy (i) and using the maximum between the output noise of $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{this work})}$ as the input noise of $\mathcal{A}^{(\text{CJP21})}$ which satisfies (ii). This approach works well for parameter couples (#1,#8) and (#2,#9). But for the last parameter set couple (#3,#10), there is no solution for $\mathcal{A}^{(\text{CJP21})}$ with the aforementioned constraints. For this special case, we reverse the order of the optimization and first solve the optimization problem for $\mathcal{A}^{(\text{CJP21})}$ and then for $\mathcal{A}^{(\text{this work})}$ with the additional constraints mentioned above.

4.3.2 Experimental results

The tables presented in this section contain timings related to 16 and 32-bit integer operations using the radix approach (Table 3), the CRT approach (Table 4) and the native CRT approach (Table 5). The benchmarks measure timings to compute homomorphic additions, multiplications, carry cleanings (apart from the native CRT approach) and LUT evaluations (only for 16-bits integers). As explained in Remark 8, it is not doable to evaluate LUT on 32-bit integers.

Radix Approach. In Table 3, dedicated to the radix approach, we display two instances of 16-bit integers and three instances of 32-bit integers. The number of additions is bounded by the room available in the carry buffer, and once it is full, a carry cleaning is needed.

For the 16-bit integers, it is possible to use both the $\mathcal{A}^{(\text{CJP21})}$ and the $\mathcal{A}^{(\text{this work})}$ operators. This means that for 16-bits integer, classical arithmetic uses the usual PBS ($\mathcal{A}^{(\text{CJP21})}$), and LUT evaluation is done with the WoP-PBS ($\mathcal{A}^{(\text{this work})}$). We assume that the WoP-PBS is done over integers with free carry buffers (i.e., after a carry cleaning). The parameters have been generated as described in Paragraph 4.3.1. Note that the addition does not require any PBS to be computed (this is denoted with a star $+^*$), but is done accordingly to the parameters generated for the PBS.

For 32-bit integers, only arithmetic operations are possible. So, cryptographic parameters are optimized following $\mathcal{A}^{(\text{CJP21})}$ only. Hence, some operations are computed faster for the 32-bit integers than for the 16-bit ones.

integer parameters				PBS based operations				WoP-PBS based operations	
Ω	p	carry modulus	κ	param ID	$+^*$	$\times$	carry cleaning	param ID	LUT evaluation
2^{16}	2^1	2^1	16	#1	12.8 µs	29.0 s	932 ms	#8	823 ms
2^{16}	2^2	2^2	8	#2	6.67 µs	5.73 s	657 ms	#9	1.80 s
2^{32}	2^1	2^1	32	#4	19.1 µs	43.8 s	685 ms	∅	
2^{32}	2^2	2^2	16	#5	12.3 µs	9.60 s	514 ms		
2^{32}	2^4	2^4	8	#6	137 µs	25.0 s	6320 ms		

Table 3: Benchmarks for 16-bit and 32-bit homomorphic integers based on the radix approach. The star (*) means that a PBS is not required to compute the operation.

Remark 12 (Multiplication Failure Probability) *When 32-bit integers are represented with 32 blocks (i.e., $\kappa = 32$), the number of AP of type $\mathcal{A}^{(CJP21)}$ required to compute a multiplication is quadratic in the number of blocks. Because the error probability p_{fail} of this AP is bounded by $2^{-13.9}$ in our experiments, the error probability at the level of the multiplication will be increased greatly. To balance this, one can use the technique described in section 5.2. Timings are clearly not in favor of this representation, and the probability of having an error is small enough for the other representations (with a smaller number of blocks). One solution is to keep the same value of p_{fail} and consider a small enough κ, resulting in a better trade-off between running time and failure probability at the multiplication level (e.g., the one associated with the parameter ID#5). Another way of solving this problem would be to have another parameter set dedicated to the multiplication algorithm, with a smaller failure probability p_{fail} but we leave that as an future work.*

CRT Approach. Table 4 is dedicated to the CRT approach. In this representation, each block have a dedicated basis, and are independent by construction. We display one instance for 16-bit integers and another one for 32-bit integers. For both of them we show the total time needed to compute the operations, as well as the amortized time when the implementation is multi-threaded. As for Table 3, the number of additions is bounded by the room available in the carry buffer, and once it is full, a carry cleaning in needed. Note that in the case of homomorphic evaluation of polynomial functions, using the CRT representation offers better timings, since it is sufficient to compute a PBS on each CRT residue. The timings are then the same as the ones of the carry cleaning when there is one block per residue, otherwise it means that we are considering the hybrid approach, and in that case, it is the cost of a LUT evaluation separately on each block.

For the 16-bit integers, the basis is given by $\Omega = 2^3 \cdot 3^2 \cdot 7 \cdot 11 \cdot 13 \approx 2^{16}$. As for 16-bit in radix representation, it is possible to use both the $\mathcal{A}^{(\text{CJP21})}$ and the $\mathcal{A}^{(\text{this work})}$ operators. However, the major difference here is about the parameter optimization: in this case, the atomic pattern $\mathcal{A}^{(\text{CJP21})}$ has been privileged. Thus, the timings for the evaluating a LUT using a WoP-PBS are way slower. By removing the constraint of compatibility, the performance should be closer to the one of Table 5. The WoP-PBS is parallelized by extracting bits for each block independently. Then, each LUT evaluation outputting one block (and taking all bits as input) is computed in parallel: note that this approach could also be applied in the case of the radix

decomposition.

We consider the basis defined by $\Omega = 2^5 \cdot 3^5 \cdot 5^4 \cdot 7^4 \approx 2^{32}$ to represent 32-bit integers using the hybrid representation. For instance, to represent integers under the modulus 7^4, we use radix-based integers with 4 blocks and a message modulus equals to 7. Thanks to the CRT representation, by using this basis multiplications can be computed with the fast bi-variate PBS approach described in 2.2.

Ω	type of execution	**PBS based operations**				**WoP-PBS based operations**	
		param ID	$+^*$	$\times$	carry cleaning	param ID	LUT evaluation
$\approx 2^{16}$	sequential	#3	8.36 µs	401 ms	251 ms	#10	23.1 s
	5 threads		1.67 µs	80.3 ms	50.2 ms		4.61 s
$\approx 2^{32}$	sequential	#7	27.6 µs	5.17 s	2400 ms	∅	
	4 threads		8.78 µs	1.82 s	729 ms		

Table 4: Benchmarks for 16-bit homomorphic integers based on the CRT approach and 32-bit integers are computed with a hybrid approach. We use the following CRT basis: $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and $\Omega = 2^5 \cdot 3^5 \cdot 5^4 \cdot 7^4 \approx 2^{32}$.

Native CRT Approach. In Table 5, dedicated to the native CRT approach, we display one instance of 16-bit integers and another of 32-bit integers. We consider $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and respectively $\Omega = 3 \cdot 11 \cdot 13 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 32 \approx 2^{32}$ Since there is no carry buffer in this representation, there is no need for a carry cleaning. However, to avoid incorrect computation, the number of additions is bounded for these parameter sets by the value ν. Once this bound is reached, a WoP-PBS is required to reduce the noise.

We observe a slower timing for the multiplication with 32-bit integers, 36.8 seconds, which leads us to think that for the precision around 32 bits, a hybrid approach is more efficient. Indeed, the native CRT approach requires to have in a single LWE ciphertext a small enough noise (after the bootstrapping) to preserve the message (with a size equal to the co-prime modulo) and the room for the 2-norm ν needed to compute multiplications with known integers or additions between ciphertexts. So when one tries to build a big Ω, since small prime numbers are not infinite, they end up with big co-prime residues and as a consequence needs big 2-norm which means very slow parameter sets.

Ω	type of execution	**WoP-PBS based operations**				
		param ID	$+^*$		$\times$	LUT evaluation
			ν	time		
$\approx 2^{16}$	sequential	#11	5	4.32 µs	7.42 s	3.81 s
	5 threads			0.862 µs	1.65 s	0.761 s
$\approx 2^{32}$	sequential	#12	2^5	6.98 µs	36.8 s	∅
	8 threads			0.873 µs	5.31 s	

Table 5: Benchmarks for 16-bit and 32-bit homomorphic integers based on the native CRT approach. We use the following CRT basis: $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and $\Omega = 3 \cdot 11 \cdot 13 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 32 \approx 2^{32}$.

Remark 13 (Timing for the Optimization) *For most of the studied AP in this paper, the parameter search has been optimized: it takes less than a second to generate a curve such as in Figures 8. The only atomic pattern taking more time to get a parameter set generated is* $\mathcal{A}^{(GBA21)}$*. Indeed, we did not implement shortcuts in it parameter search resulting in 26 minutes to generate a curve.*

5 An Optimization Framework for FHE

In this section, we introduce the full fledged problem we want to ultimately tackle. We also show how to adapt our framework to work with the probability of failure for the whole graph and not only the probability of failure of atomic operations. We will then expand the comparison between atomic pattern types and explain other key features of our optimization framework. For instance, we are able to optimally insert bootstrapping in leveled operations to decrease the noise, but we are also able to deal with several public keys (bootstrapping and key switching keys). We will also show how to tweak an atomic pattern to obtain a consensus-friendly version of TFHE.

5.1 Full-Fledged Problem

We introduced in definition 7 the notion of FHE DAG. Such structure is filled with nodes symbolizing an FHE operator or a sub-graph of FHE operators. In real life situation, one owns a graph of computation and wants to deploy an FHE scheme to compute the same graph but over encrypted data. It means that the problem we eventually address in this paper is way more complicated that what we previously explained in section 3. Instead of taking a graph of FHE Operators, we take as input a crypto-free graph and find at the same time, the best FHE DAG and its associated cryptographic parameters which guarantee that it behaves as the input DAG but over encrypted inputs. The following definition formalizes this notion.

Definition 13 (Plain DAG) *Let $\overline{\mathcal{G}} = \left(\overline{V}, \overline{E}\right)$ be a DAG of plain operators. We define $\overline{V} = \left\{\overline{\mathcal{O}}_i\right\}_{1 \leq i \leq \alpha}$ as the set of vertices, each of them being a plain operator that can be additions, multiplications, subtractions, LUT evaluation, and many others. We define $\overline{E}$ as the set of edges, each of them associated with the precision p of the message as well as a label which is either "private" or "public". When $\overline{E}$ is not needed, we will simply write $\overline{\mathcal{G}} = \overline{V}$.*

We note $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right)$ the set of all possible FHE graphs computing the same functionality than the plain DAG $\overline{\mathcal{G}}$.

Our optimization framework takes as input a plain DAG $\overline{\mathcal{G}}$, a level of security, and a correctness probability. It outputs an FHE DAG $\mathcal{G}$ as well as a parameter set x for $\mathcal{G}$. Remember most of the FHE operators in $\mathcal{G}$ introduce some cryptographic parameters, for instance a local polynomial size $N \in \mathbb{N}$ or a local base $\beta \in \mathbb{N}$. It implies that the total number of possible parameter sets is exponentially huge and we want x to be the best of them all. Also remember that for a same plain operator, for instance a homomorphic multiplication, there are many possible strategies to translate it into an FHE sub-graph. In this example, we could use the leveled multiplication in the TFHE-context as explained in [CLOT21] or two PBS as in [CJL+20]. Those different ways to translate a plain operator into an FHE one make the problem more complex as the size of $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right)$ depends on the list of known translation rules between plain and FHE operators.

The output of our optimization framework, an FHE DAG $\mathfrak{G}$ and its associated parameter set x must fulfill guaranties 1, 2, 3 and a last one:

4. the FHE DAG computes the plain functionality described in the plain DAG.

The full-fledged problem we want to solve is the following:

$$\left(\widehat{\mathfrak{G}}, \widehat{x}\right) = \text{argmin}_{\mathfrak{G},x} \mathsf{Cost}\left(\mathfrak{G}, x\right) \;\; s.t. \;\; \mathfrak{G} \in \mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathfrak{G}}\right) \;\; x \in \mathcal{S}\left(\mathfrak{G}\right) \tag{4}$$

To build $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathfrak{G}}\right)$, we use simple translation rules between plain operators and FHE operators (definition 4). To compute a lookup table or a function, we can use a KS and a PBS from [CGGI20] if the precision of the message is between 1 and 8 bits or the WoP-PBS described in algorithm 3 for larger precisions. A multiplication between ciphertext can be replaced by two PBS from [CGGI20] and an addition as described in [CJL+20] (end of section 2.). A DP can be replaced by the same DP working over ciphertexts. We will explain in 5.3 how to do a better transformation by optimally inserting PBS in the DP.

5.2 Failure Probability: From the AP to the Entire Graph

In the previous section, all failure probabilities were associated with one single FHE operator (at least those where there is effectively a risk). We want to extend our framework to work directly with the probability of failure of the entire graph. To do so, we start by exposing a simpler case where two AP are compared, before generalizing the method to a whole graph.

Observe that it is easy to have an upper bound on the graph failure probability given individual AP failure probabilities. Let $\mathfrak{G} = \{A_i\}_{i \in I}$ be an FHE DAG and let's assume, without loss of generality, that $\mathfrak{G}$ has an unique output $\mathsf{ct}\left(\widetilde{m_{\mathsf{out}}}\right)$. For every $i \in I$, let $\mathsf{ct}\left(\widetilde{m_i}\right)$ be the ciphertext for which the noise is the highest in A_i. Than, the failure probability for the whole graph is bounded by:

$$p_{\mathsf{fail}}\left(\mathfrak{G}\right) = \mathbb{P}\left(\mathsf{Decode}\left(\mathsf{ct}\left(\widetilde{m_{\mathsf{out}}}\right)\right) \neq m_{\mathsf{out}}\right) \leq 1 - \prod_{i \in I}\left(1 - p_{\mathsf{fail}(A_i)}\right) \tag{5}$$

with $p_{\mathsf{fail}}\left(A_i\right) = \mathbb{P}\left(\mathsf{Decode}\left(\mathsf{ct}\left(\widetilde{m_i}\right)\right) \neq m_i\right), \forall i \in I$. By applying the *domination* concept presented in Theorem 1, when an atomic pattern A_1 is dominated by an atomic pattern A_2, we can find a relationship between $p_{\mathsf{fail}}\left(A_1\right)$ and $p_{\mathsf{fail}}\left(A_2\right)$.

Let α be the failure probability that we want to guarantee for every atomic pattern, and let $\kappa = z^*\left(\alpha\right)$ be its associated standard score. Let $(\nu_1, \nu_2) \in \mathbb{R}^2$ s.t. $\nu_1 \leq \nu_2$, and let (p_1, p_2) be two precisions s.t. $p_1 \leq p_2$, which means that $t_1 = \frac{q}{2^{1+p_1+1}\cdot\kappa} = \frac{\Delta_1}{2\kappa} \geq t_2 = \frac{q}{2^{1+p_2+1}\cdot\kappa} = \frac{\Delta_2}{2\kappa}$. Let $A_1 = A\left(\nu_1, t_1\right)$ and $A_2 = A\left(\nu_2, t_2\right)$ be two atomic patterns of type $\mathcal{A}^{(\text{CJP21})}$.

As $\nu_1 \leq \nu_2$ and $t_1 \geq t_2$, A_1 is dominated by A_2 (Theorem 1). It means that if we have $|e_2| < \frac{\Delta_2}{2}$, we know that $|e_1| < \frac{\Delta_1}{2}$ with $e_1 \hookleftarrow \mathcal{N}\left(0, \sigma_1^2\right)$ (respectively $e_2 \hookleftarrow \mathcal{N}\left(0, \sigma_2^2\right)$) is the maximal noise in A_1 (respectively A_2).

Following Definition 8, we know that $\sigma_2 \leq t_2 \Rightarrow \mathbb{P}\left(|e_2| \geq \frac{\Delta_2}{2} = \kappa \cdot t_2\right) \leq \alpha$. It follows that, if this inequality is met, we will also have $\mathbb{P}\left(|e_1| \geq \frac{\Delta_1}{2}\right) = p_{\mathsf{fail}}(A_1) \leq \alpha$ (Theorem 1).

At this point, we look for an estimation of the failure probability $p_{\mathsf{fail}}(A_1)$ as a function of $p_{\mathsf{fail}}(A_2)$. The noise inside an atomic pattern of type $\mathcal{A}^{(\mathrm{CJP21})}$ is maximal after the modulus switching. With our noise model we have that:

$$\forall i \in I, \sigma_i^2 = \sigma_{\mathsf{BR}}^2 \cdot \nu_i^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2$$

With $\sigma_2 \leq t_2$ and $\sigma_1 \leq t_1$, we have:

$$\sigma_1^2 \leq t_2^2 - (\nu_2^2 - \nu_1^2) \cdot \sigma_{\mathsf{BR}}^2 \leq t_1^2$$

With the previous inequality, we have found a tighter noise bound than before and we can compute its associated standard score κ_1.

Let's assume that there exists a real number $D \in \mathbb{R}^*$ such that, for every possible set of parameter x, $\sigma_{\mathsf{BR}}^2(x) \geq D$, i.e., $D = \min_x \sigma_{\mathsf{BR}}^2(x)$. Using the previous inequality, we have:

$$\begin{aligned}
\sigma_1^2 &\leq t_2^2 - (\nu_2^2 - \nu_1^2) \cdot \sigma_{\mathsf{BR}}^2 \\
&\leq t_2^2 - (\nu_2^2 - \nu_1^2) \cdot D = t_{1,\mathsf{new}}^2 \\
&= \left(\frac{\Delta_1}{2 \cdot \kappa_1}\right)^2 \leq t_1^2
\end{aligned}$$

and so we have:

$$\kappa_1 = \frac{\Delta_1}{2} \cdot \left(D \cdot (\nu_1^2 - \nu_2^2) + \left(\frac{\Delta_2}{2\kappa}\right)^2\right)^{-\frac{1}{2}}. \tag{6}$$

Using Definition 1, we have $p_{\mathsf{fail}}(A_1) \leq 1 - \mathsf{erf}\left(\frac{\kappa_1}{2}\right)$. To find an adequate D, one can iterate over every possible set of parameters x and find the minimal value for $\sigma_{\mathsf{BR}}(x)$. In particular, if $\nu_1 = \nu_2$, we have $\kappa_1 \geq \frac{\Delta_1}{\Delta_2} \cdot \kappa$.

Using the relationship above, we have a simple algorithm to find parameters that satisfy a given failure probability for a whole graph $\mathcal{G} = \{A_i\}_{i \in I}$. Let $p_{\mathsf{fail}}(\mathcal{G})$ be the failure probability we want to guarantee for the whole graph, and let δ_{fail} be its associated tolerance. The algorithm will output a failure probability $p_{\mathsf{fail}}(A)$ that can be used as described in the sections above and we are sure to achieve a failure probability $p_{\mathsf{fail}}(\mathcal{G})$ such that $\left|p_{\mathsf{fail}}(\mathcal{G}) - p_{\mathsf{fail}}(\mathcal{G})\right| < \delta_{\mathsf{fail}}$.

First, we build $\mathcal{G}_{\mathsf{pareto}} = \{A_i\}_{i \in I'} \leftarrow \mathsf{Pareto}(\mathcal{G})$, as defined in Section 3.2. Then, we set $p_{\mathsf{fail}}(A_{\mathsf{dominant}}) \leftarrow 1 - (1 - p_{\mathsf{fail}}(\mathcal{G}))^{\frac{1}{|I'|}}$. At this stage, we can apply what is described above to find the probability of failure of the dominated atomic patterns i.e. compute $\forall i \in I \setminus I', p_{\mathsf{fail}}(A_{\mathsf{dominated},i})$. Then, using equation 5, we can compute

$$\widetilde{p_{\mathsf{fail}}(\mathfrak{G})} \approx 1 - (1 - p_{\mathsf{fail}}(A_{\mathsf{dominant}}))^{|I'|} \cdot \prod_{i \in I \setminus I'} (1 - p_{\mathsf{fail}}(A_{\mathsf{dominated},i}))$$

If $\left|\widetilde{p_{\mathsf{fail}}(\mathfrak{G})} - p_{\mathsf{fail}}(\mathfrak{G})\right| > \delta_{\mathsf{fail}}$, we need to decrease or increase $p_{\mathsf{fail}}(A_{\mathsf{dominant}})$ and to repeat the rest of the algorithm until we meet the condition.

5.3 Optimal PBS Insertion

In Section 5.1, we suggested to translate a plain DP into an FHE DP. Here, we explain how to automatically insert PBS during a DP wherever it is interesting with regards to the cost model. It could sound counter intuitive as the PBS can be a very costly operator, hence inserting PBSs will increase the total cost of the computation. However when the 2-norm of a DP operator is high, the parameters must be large enough to still guarantee the correctness of the computation and those large parameters will have an impact on the cost. We need our framework to choose whether it is interesting to split the DP operator or not to end up with more PBSs but with smaller cryptographic parameters.

The following theorem explores this approach.

Theorem 2 (DP Splitting) *Let $A(\nu, t)$ be an AP of type $\mathscr{A}^{(CJP21)}$ with a noise bound t and including a DP of 2-norm ν. Let $\widetilde{A}(\nu, t, d)$ the same AP than A but where its DP is split into $d+1$ sub-DP of approximately the same 2-norm and connected together with PBS. It actually breaks an AP into $d+1$ AP of the same type organised in two layers (d followed by a last one connecting them all).*

Let $\mathfrak{G} = \{A(\nu_i, t)\}_{0 \leq i < Y}$ such that $\nu_0 < \nu_1 < ... < \nu_{Y-1}$. Let $\vec{d^*} = (d_0^*, \cdots, d_{Y-1}^*)$ and $\vec{d} = \left(d'_0, \cdots, d'_{Y-1}\right)$ be two different possible splitting solutions. We define the following two FHE graphs: $\mathfrak{G}^* = \left\{\widetilde{A}(\nu_i, t, d_i^*)\right\}_{0 \leq i < Y}$ and $\mathfrak{G}' = \left\{\widetilde{A}(\nu_i, t, d'_i)\right\}_{0 \leq i < Y}$.

If every coordinates of d^* is inferior or equal (coordinate wise) to the ones from $\widetilde{d}$ and $\mathcal{S}(\mathfrak{G}^*) = \mathcal{S}\left(\widetilde{\mathfrak{G}}\right)$, then, $\widetilde{d}$ cannot be the optimal solution.

Proof 6 (Sketch) *Let $x \in \mathcal{S}(\mathfrak{G}^*) = \mathcal{S}\left(\widetilde{\mathfrak{G}}\right)$, we have $\mathsf{Cost}(\mathfrak{G}^*, x) < \mathsf{Cost}(\mathfrak{G}', x)$ as there are more atomic patterns in $\mathfrak{G}'$ than in $\mathfrak{G}^*$ so $\mathfrak{G}'$ cannot be the result of equation 4.* □

Application to AP of type $\mathscr{A}^{(\mathbf{CJP21})}$

We consider a graph $\mathfrak{G} = \{A(\nu_i, t)\}_{0 \leq i < \alpha}$ composed of AP of type $\mathscr{A}^{(\mathrm{CJP21})}$ which involves a DP operator. We introduce a new AP type and translate every AP of type $\mathscr{A}^{(\mathrm{CJP21})}$ into this new AP type $A^* \in \mathscr{A}^{(\mathrm{CJP21}^*)}$ which has the exact same sub-graph than AP type $\mathscr{A}^{(\mathrm{CJP21})}$ except that the DP is split into several sub-DP connected with PBS as explained in Theorem 2. This AP has a new parameter d_i describing the splitting of the DP for the i-th AP, i.e., how many sub-DP we will have. Note that a graph $\mathfrak{G}^*$ with fixed values d_i can be viewed as a graph $\widetilde{\mathfrak{G}^*}$ of AP type $\mathscr{A}^{(\mathrm{CJP21})}$.

Formally, equation 4 can be written this way:

$$\left(\widehat{\mathcal{G}}, \widehat{x}\right) = \underset{\vec{d}\in\mathcal{D}}{\arg\min}\, \mathsf{Cost}\left(\mathcal{G}^*_{\vec{d}}, x^*\right) \;\; s.t. \;\; \mathcal{G}^*_{\vec{d}} = \left\{A^*(\nu_i, t, d_i)\right\}_{0\leq i<\alpha} \qquad (7)$$
$$x^* \text{ solution of equation 2}$$

Several ways can be imagined to split a DP. One could be to group public weights of the DP into d_i sets such that their 2-norm is approximately the same. This will yield the best result if we keep neglecting the cost of the DP. Inserting a PBS adds extra operations to perform, but it will also reduce the 2-norm of the initial DP.

To find how to split in two a dot product with a 2-norm ν and with the weights $\{\omega_i\}_{i\in J}$, we can solve the following problem

$$\begin{aligned} \min_{\omega_{i,1},\Lambda} \max(\nu_1, \nu_2) \text{ s.t. } & \omega_i = \omega_{i,1}\Lambda + \omega_{i,2} \\ & \mathsf{GCD}\left(\omega_{i,1}, \Lambda\right) = 1 \\ & \forall j \in \{1,2\}, \nu_j = \sqrt{\sum_i \omega_{j,1}^2} \end{aligned} \qquad (8)$$

This will yield two sets of weights $\{\omega_{i,1}\}_{i\in J}$ and $\{\omega_{i,2}\}_{i\in J}$ with $\nu_1^2 = \sum_{i\in J}\omega_{i,1}^2 \approx \nu_2^2 = \sum_{i\in J}\omega_{i,2}^2$.

If we don't know the weights $\{\omega_i\}_{i\in J}$ but we know that they follow an uniform distribution between -2^p and 2^p, we have a trivial way yet very efficient (regarding the noise) to split a DP in $d+1$ DP. We can radix-decompose all the DP weights with the level being equal to $d+1$ and the $\log_2$ of the base β is equal to $\frac{p+1}{d+1}$. Here each Λ_i is equal to a power of β.

5.4 Study of Key Switching Position

In some contexts it is possible to analytically compare two AP types prior to the optimization, i.e. for all suitable set of parameters, one of the AP is always better than the other. For instance, in the gate bootstrapping described in [CGGI20], an unlimited number sequences of PBS, KS and DP is described. However one can analytically prove that it is always best to have the KS right before the PBS. We introduced a new atomic pattern type $\mathcal{A}^{(\text{CGGI20})}$ composed of a key switch, a DP and a PBS and we compare it to $\mathcal{A}^{(\text{CJP21})}$. The following theorem formalizes the comparison between those two types.

Theorem 3 (Relation Between $\mathcal{A}^{(\mathbf{CJP21})}$ and $\mathcal{A}^{(\mathbf{CGGI20})}$) *We consider two 2-norms $\nu_1, \nu_2 \in \mathbb{R}^+$ such that $\nu_1 \leq \nu_2$, two noise bounds $t_1, t_2 \in \mathbb{N}$ such that $t_2 \leq t_1$ and two AP: $A_1 \in \mathcal{A}^{(CJP21)}$ and $A_2 \in \mathcal{A}^{(CGGI20)}$. We have $\mathcal{S}\left(A_2\left(\nu_2, t_2\right)\right) \subseteq \mathcal{S}\left(A_1\left(\nu_1, t_1\right)\right)$.*

Proof 7 *Let's start with the observation that $\mathcal{A}^{(CJP21)}$ and $\mathcal{A}^{(CGGI20)}$ share the same search space $\mathcal{P}$ because they are built with the same operators. For a parameter*

set $x \in \mathcal{P}$, the maximum noise variance in an AP of type $\mathcal{A}^{(CJP21)}$ is $\sigma^2_{\mathsf{out},1}(x) = \sigma^2_{\mathsf{in}}(x) \cdot \nu^2 + \sigma^2_{\mathbf{KS}}(x) + \sigma^2_{\mathbf{MS}}(x)$ where $\sigma^2_{\mathbf{KS}}(x)$ is the noise added by the KS and $\sigma^2_{\mathbf{MS}}(x)$ is the noise added by the MS. Similarly, the maximum noise variance in an AP of type $\mathcal{A}^{(CGGI20)}$ is $\sigma^2_{\mathsf{out},2}(x) = (\sigma^2_{\mathsf{in}}(x) + \sigma^2_{\mathbf{KS}}(x)) \cdot \nu^2 + \sigma^2_{\mathbf{MS}}(x)$.

We consider $\bar{x} \in \mathcal{S}(A_2(\nu_2, t_2))$, so we have $\sigma^2_{\mathsf{out},2}(\bar{x}) = (\sigma^2_{\mathsf{in}}(\bar{x}) + \sigma^2_{\mathbf{KS}}(\bar{x})) \cdot \nu_2^2$ and $\sigma^2_{\mathsf{out},2}(\bar{x}) < t_2^2$. The simplest non-trivial DP possible is when we only have one input ciphertext multiplied by 1, so we have $1 \leq \nu$. Since variances are positive and $1 \leq \nu_1 \leq \nu_2$, we have:

$$t_1^2 \geq t_2^2 > \sigma^2_{\mathsf{out},2}(\bar{x}) = \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathbf{KS}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathbf{MS}}(\bar{x})$$
$$\geq \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathbf{KS}}(\bar{x}) + \sigma^2_{\mathbf{MS}}(\bar{x}) \geq \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_1^2 + \sigma^2_{\mathbf{KS}}(\bar{x}) + \sigma^2_{\mathbf{MS}}(\bar{x}) = \sigma^2_{\mathsf{out},1}(\bar{x})$$

So we have $t_1^2 \geq \sigma^2_{\mathsf{out},1}(\bar{x})$, meaning that $\bar{x} \in \mathcal{S}(A_1(\nu_1, t_1))$, so $\mathcal{S}(A_2(\nu_2, t_2)) \subseteq \mathcal{S}(A_1(\nu_1, t_1))$. □

The same result can be found by solving equation 2 for atomic pattern types $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$. The curves are shown on Figure 10 for precision up to 11 bits and for four different 2-norm ν. In this figure, we also added the comparison with $\mathcal{A}^{(\text{KS-free})}$, the atomic pattern type composed of a DP and a bootstrapping from [CGGI20] (without any key switch). The cost of this AP type is plotted as the green/▼ curve the cost of $\mathcal{A}^{(\text{KS-free})}$. The blue/• curve represents the cost of $\mathcal{A}^{(\text{CJP21})}$ and the red/+ curve is the cost of $\mathcal{A}^{(\text{CGGI20})}$.

As we can see, the smallest cost is the one of $\mathcal{A}^{(\text{CJP21})}$ which confirms what we found theoretically by comparing $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$ in Theorem 3. We also notice that for precisions larger than 8 bits, there are no feasible parameters for $\mathcal{A}^{(\text{CGGI20})}$. This is due to the fact that the noise of the key switch is amplified by the DP in $\mathcal{A}^{(\text{CGGI20})}$ whereas it is not in $\mathcal{A}^{(\text{CJP21})}$. On the contrary, for very small precisions, $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$ are very similar in term of efficiency. The difference increases as soon as we increase the 2-norm factor ν. It means that for Boolean TFHE (gate bootstrapping described in [CGGI20]), having the key switching after the bootstrap does not worsen the cost by much, but it will not stay that way for larger precisions. This figure also illustrates the usefulness of the key switching as $\mathcal{A}^{(\text{KS-free})}$ is always the worst atomic pattern type in term in cost. Furthermore, for $\mathcal{A}^{(\text{KS-free})}$ no solution is found for precisions larger than 7 bits. To conclude, one always wants to compute the KS right before the PBS as in AP of type $\mathcal{A}^{(\text{CJP21})}$.

Remark 14 (Mixing Different AP Types in an FHE Graph) *We consider an FHE graph $\mathcal{G}$ containing two types of AP: type $\mathcal{A}^{(CJP21)}$ and type $\mathcal{A}^{(CGGI20)}$. We can apply the AP domination (Theorem 1) on every atomic pattern of each type. At the end of this procedure, we end up with a few AP of type $\mathcal{A}^{(CJP21)}$ and a few AP of type $\mathcal{A}^{(CJP21)}$. To further simplify the problem, we can use theorem 3 to compare the remaining atomic patterns of $\mathcal{A}^{(CJP21)}$ with the atomic patterns of $\mathcal{A}^{(CGGI20)}$.*

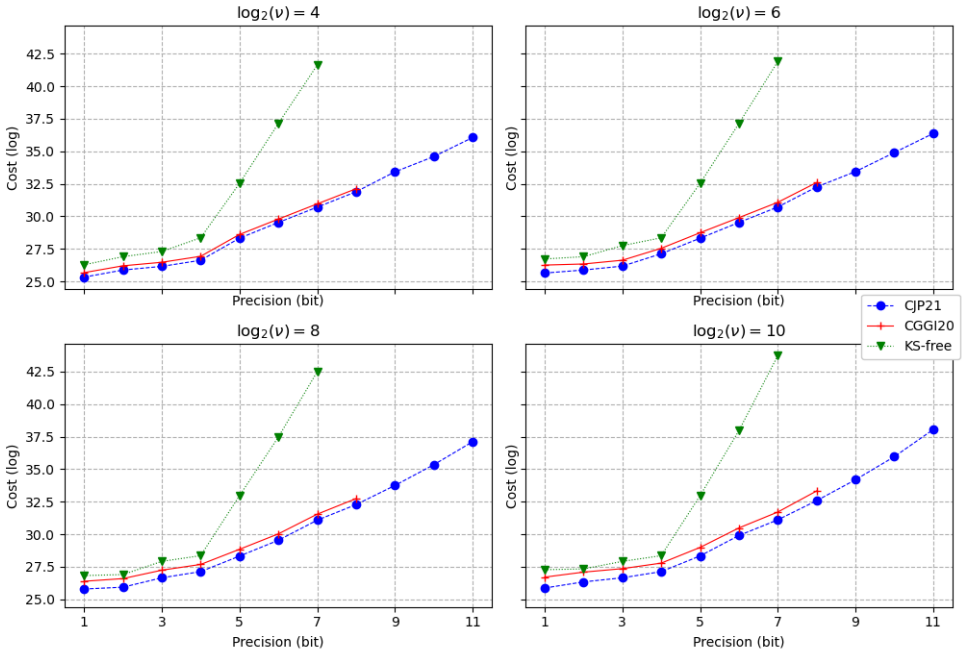

Figure 10: In this figure, we compare AP of type $\mathcal{A}^{(\text{CJP21})}$, type $\mathcal{A}^{(\text{CGGI20})}$ and type $\mathcal{A}^{(\text{KS-free})}$.

5.5 Failure Probability Spectrum

In this section we analyze the impact of decreasing the failure probability on the cost, for the atomic patterns $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{this work})}$. We consider four different failure probabilities: $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$ and to simplify the analysis, we fix the 2-norm $\nu = 2^4$ since the behaviour is pretty similar for other 2-norms.

Figure 11 is dedicated to AP of type $\mathcal{A}^{(\text{CJP21})}$. We plot the cost for precisions between 1 and 12 bits. As expected, we can observe that if we decrease the failure probability, the cost increases. Roughly speaking, starting from 4-5 bits of precision, for every additional bit, N has to be twice as big, which doubles the cost of the atomic pattern. Observe that the cost is very close for certain curves: as instance, the brown/◆ curve, corresponding to $p_{\mathsf{fail}} = 2^{-50}$, and the green/▼ curve, corresponding to $p_{\mathsf{fail}} = 2^{-35}$, have almost the same cost. The red/+ curve, corresponding to $p_{\mathsf{fail}} = 2^{-20}$, has a cost that is close to the one of the blue/• curve, corresponding to $p_{\mathsf{fail}} = 2^{-14}$, up to 7 bits of precision, and starting from 8 bits of precision it gets closer to the green/▼ curve. This change is due to the fact that there are no parameter sets fulfilling the requirements with a bigger N only twice as big, it has to be 4 times bigger.

Figure 12 is dedicated to AP of type $\mathcal{A}^{(\text{this work})}$. The brown/◆ curve, corresponds to $p_{\mathsf{fail}} = 2^{-50}$, the green/▼ curve, corresponds to $p_{\mathsf{fail}} = 2^{-35}$, the red/+ curve corresponds to $p_{\mathsf{fail}} = 2^{-20}$, and finally the blue/• curve corresponds to $p_{\mathsf{fail}} = 2^{-14}$. All the curves follow the same behaviour and are simply shifted up when the failure

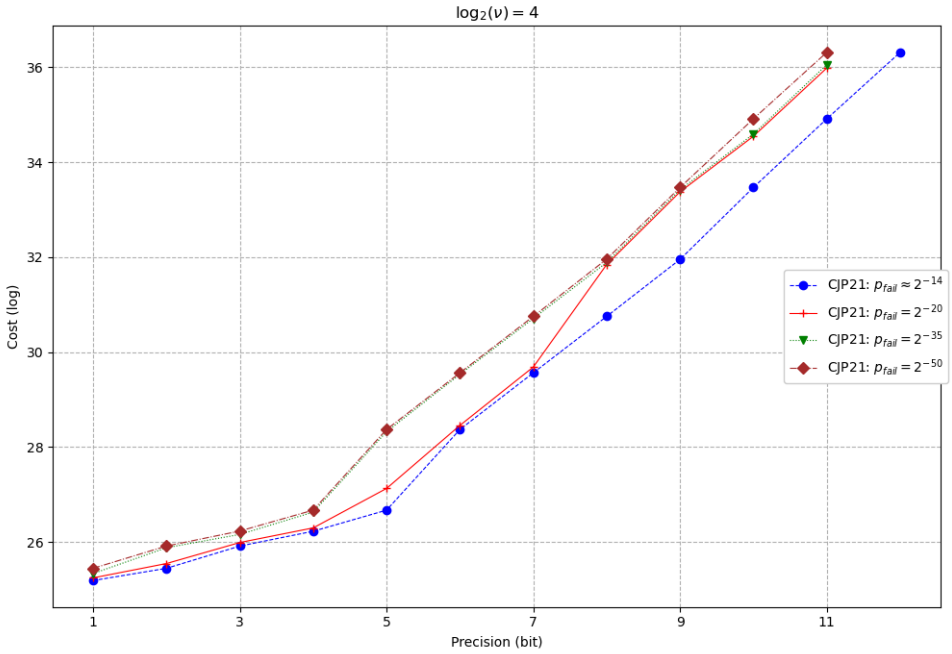

Figure 11: Cost comparison for the same AP of type $\mathcal{A}^{(\text{CJP21})}$, with respect to the following failure probabilities: $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$.

probability is decreased.

To sum up, with the AP of type $\mathcal{A}^{(\text{CJP21})}$, for each additional bit of precision, the overall cost of the AP is doubled. However, this is not the case with the AP of type $\mathcal{A}^{(\text{this work})}$. The behaviour of the curve in this region (precision below 24 bits) looks more like a linear one. For probabilities $p_{\mathsf{fail}} = 2^{-35}$ and $p_{\mathsf{fail}} = 2^{-50}$, the curves are almost overlapping. If we look for instance at 7 bits of precision, the polynomial size N are the same, but the noise added by the key switch is slightly lower thanks to a bigger n (output of the key switch) and to other small changes in the key switch decomposition parameters. Indeed, at this precision, N is already quite big so it can handle the message precision. Thus the noise of the modulus switch is not the most constraining one, the one from the key switch actually is. This explains the small overhead in this context.

5.6 Optimization for Several Public Keys

In previous results on atomic pattern type $\mathcal{A}^{(\text{CJP21})}$, we assumed that we have only one public material per FHE operator for the whole FHE DAG as we were only looking for one polynomial size, one GLWE dimension and one LWE dimension.

Restricting the number of public keys helps to have a small quantity of public material. It also has an impact on the complexity of the optimization problem because as a result, parameters are shared across the entire FHE DAG. The down side is that we cannot speed up parts of the FHE DAG that have a bigger noise bound or less leveled operations (smaller 2-norm) with smaller and faster parameters.

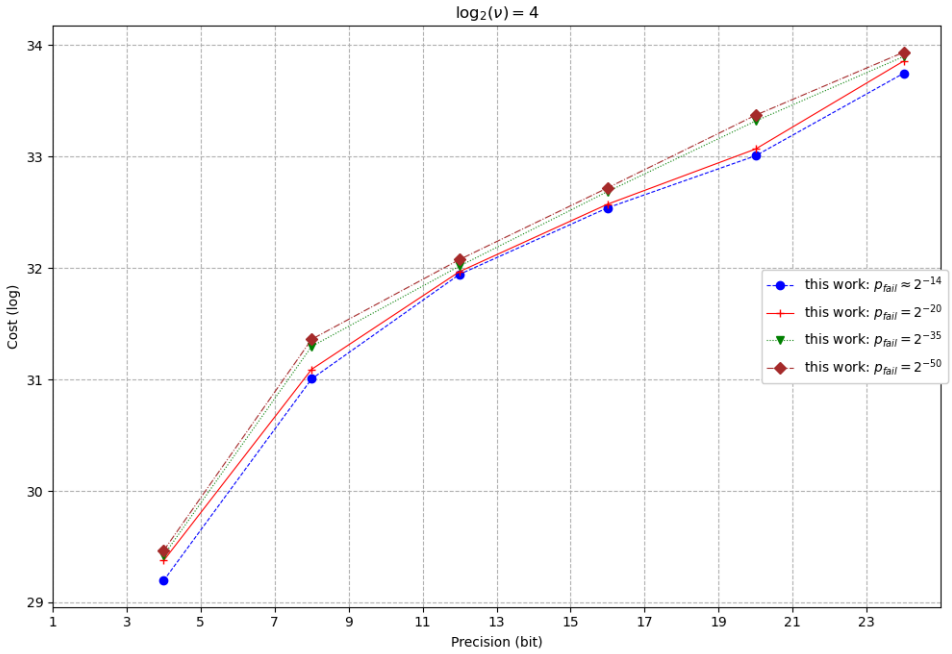

Figure 12: Cost comparison for the same AP of type $\mathcal{A}^{(\text{this work})}$, with respect to the following failure probabilities: $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$.

In this section we describe a simple optimization problem: one LWE secret key $\vec{s}$ and one GLWE secret key $\vec{S}'$ (for the PBS) that can be viewed as a bigger LWE secret key $\vec{s}'$, along with X different key switching keys going from $\vec{s}'$ to $\vec{s}$. These key switching keys can use a different base β and/or a different number of levels ℓ. We consider a graph $\mathcal{G}$ of Y APs of type $\mathcal{A}^{(\text{CJP21})}$.

There are many ways to solve this problem. A naive solution is to consider different parameters for each KS and to let every KS to have its own (β, ℓ) and to add the constraint that they can have at most X values. This approach increases exponentially the search space of the optimization problem, so we will not consider it.

A second solution, which is straightforward though not the most efficient one, is to introduce a new variable δ for each KS. This value δ stores the associated KSK identifier. This is a new parameter to optimize for each KS: $\forall i \in [1, Y]\, \delta_i \in [1, X]$. So our additional search space, defined by the problem of finding which KSK is used by which KS, is of size X^Y.

We designed a third solution to solve the problem. Starting now, we will sort our KSK $(\mathsf{KSK}_0, \mathsf{KSK}_1, \cdots)$ such that a KS with KSK_i adds less noise than using KSK_{i+1} for all $0 \leq i$. Ranking the noise added by the key switching keys allows to use the following theorem.

Theorem 4 (Optimal KSK) *Let* KSK_0 *and* KSK_1 *two KS keys obtained through the resolution of the optimization problem. W.l.o.g., let us assume that a KS using*

KSK_0 *adds strictly less noise than the one using* KSK_1*, then the KS with* KSK_0 *will be slower than the KS with* KSK_1*.*

Proof 8 *The two keys must have different parameters for the base and/or the number of level, because the noise added is different by hypothesis. If the optimization has selected two distinct keys it means that they both satisfy a different cost/noise trade-off. Then, if a KS with* KSK_1 *is slower than a KS with* KSK_0 *and generates more noise,* KSK_1 *will always be worse (both in terms of noise and cost) than* KSK_0 *which contradict the fact that the optimization has selected two distinct keys.* □

Let's consider the following toy example: $t \in \mathbb{N}$ is a noise bound and $\mathcal{G} = \{A_0(\nu_0, t), A_1(\nu_1, t), A_2(\nu_2, t)\}$ is an FHE DAG such that $\nu_0 < \nu_1 < \nu_2$. This graph has the same noise bound t for each AP and they are all of type $\mathcal{A}^{(\text{CJP21})}$. Let us have two possible key switching keys and let us assume that $\vec{\delta} = (\delta_0, \delta_1, \delta_2) = (0, 1, 0)$ is the optimal solution i.e. we use KSK_0 for A_0 and A_2, KSK_1 for A_1. We will show that it cannot be so. We can use theorem 1 to infer that $\mathcal{S}(A(\nu_2, t)) \subseteq \mathcal{S}(A(\nu_1, t)) \subseteq \mathcal{S}(A(\nu_0, t))$, and Theorem 4 to infer that a KS with KSK_1 is faster than a KS with KSK_0. It is then straightforward to see that if $(0, 1, 0)$ is a solution, then $(1, 1, 0)$ (KSK_0 for A_2 and KSK_1 for A_0 and A_1) is also a solution but a faster one. This example can be extended to an arbitrary number of AP sharing the same noise bound and for an arbitrary number of key switching keys as described in the theorem 5 below.

Theorem 5 (Several KSK) *Let* $\mathcal{G} = \{A(\nu_i, t)\}_{0 \leq i < Y}$ *an FHE graph only composed of AP type* $\mathcal{A}^{(CJP21)}$ *such that* $\nu_0 < \nu_1 < \nu_2 < \cdots < \nu_Y$*. We consider that we can have* X *KSK. The optimal* $\vec{\delta} = (\delta_0, \cdots, \delta_{Y-1})$ *has the property that for all* $0 \leq i < Y - 1$ *there is* $\delta_i \geq \delta_{i+1}$*.*

Proof 9 (Sketch) *Following the same logic as in the toy example above, it is easy to prove this theorem.* □

Using the theorem 5, we can solve the optimization problem without considering every $\vec{\delta}$ that cannot be optimal solution of Equation 2.

We can consider an FHE DAG with different noise bounds, and apply what we just described for each of the different noise bounds. The same approach works to enable several BSK in the optimization. Indeed, BSK can also be sorted by amount of noise they offer in their output ciphertexts.

5.7 Consensus-Friendly TFHE & Blockchain Application

Two implementations of the same FHE algorithm that does not involve the FFT will output the same result as long as it operates over the same inputs (same ciphertexts and same public materials). For instance, different implementations of a DP or an LWE-to-LWE KS will produce the same outputs.

However, implementations that leverage the FFT output different ciphertexts depending on the FFT algorithm involved. To highlight this, we made an experiment with the traditional parameter set of TFHE-lib for the bootstrapping. We use the same secret keys, the same bootstrapping key and the same input ciphertexts, but two different implementations of the PBS with their respective FFT implementations.

We computed the difference on the resulting ciphertexts for the two different implementations, we call this value the *error* of the ciphertexts, which is different from the noise needed for security in the plaintext. We observed that the ciphertexts had the same most significant bits but their least significant bits were different. We also re-run the experiment with different parameters: more levels and bigger polynomials in the bootstrapping key. The messages encrypted were still correct but the ciphertexts were completely different.

From those experiments, we can conclude that for a given parameter set and a ciphertext with a given input error, the PBS with a given FFT either resets the error to a minimal level or outputs the maximum amount of error, i.e., a re-randomization of the ciphertext. It means that an FHE circuit containing a DP, a KS and a PBS will not output the exact same ciphertext if it is run on the same inputs with different implementations. This is not compatible with use-cases where one actually needs to guarantee reproducibility across different implementations.

Thankfully, it is possible to ensure the reproducibility by tweaking a bit our optimization framework as well as the PBS algorithm. The idea is to use a new AP type that is identical to type $\mathcal{A}^{(\text{CJP21})}$ but with an extra rounding step at the end (right after the PBS). This rounding procedure aims to remove the LSB of the ciphertexts that are different from an implementation to the other. This rounding increases the noise in the plaintext and it adds a new parameter to optimize: the location of the rounding. The higher in the MSB we round the more noise we add, but also the more error we remove. Note that this rounding will either keep the same amount of error (when the output error is maximal, i.e., the ciphertexts are completely different) or cancel it entirely depending on the parameter for the rounding and the input error.

We can add to the optimization framework a new constraint related to the maximum error of the FFT we want to consider. We can do that easily with an additional feasible set $\mathcal{S}_{\mathsf{other}}(\mathcal{G})$. The condition could be represented as: $\mathcal{S}_{\mathsf{other}}(\mathcal{G}) = \{x \in \mathcal{P} | \forall i, \mathbb{E}_{\mathsf{PBS}_i}(x) = 0\} \subset \mathcal{P}$, with $\mathbb{E}_{\mathrm{PBS}_i}(x)$ the error of the output ciphertext coefficients of a bootstrapping after the rounding. This approach relies on the fact that we have a model for the error in the output of the PBS in terms of ciphertext coefficients.

Some cryptographic observations can help reducing the size of the parameter space. In particular, we must have: $\frac{q}{2\beta_{\mathsf{PBS}}^{\ell_{\mathsf{PBS}}}} > \mathsf{error}_{FFT}(x)$.

This optimization enables to set a limit in terms of error in the ciphertext coefficients that an implementation of the FFT introduces. Then, we can optimize for a given FFT error model, some noise model, and some cost model targeting a common architecture for instance. The result of the optimization can be used to

compute the same circuit on the same ciphertexts with the same public keys, but with different implementation of the same FHE algorithms and we will end up with the exact same ciphertext in output.

This feature enables many miners in a blockchain for instance, to compute the same circuit on the same inputs and have a consensus without a need to decrypt anything. It guarantees that the result came out of the desired FHE DAG and not another designed by an attacker.

6 Conclusion & Future Work

Finding parameters that are correct, secure and efficient is a hard problem that hinders large scale adoption of FHE. In this paper, we proposed the first optimization framework that allows us to efficiently select the best FHE parameters for TFHE-like schemes given a *plain graph*, a cost model and a noise models.

In this paper, we proposed new types of ciphertexts, combining several LWE encryptions to encode large precision messages, by extending radix-based and CRT-based representations into a new hybrid representation taking the best of both worlds. We introduced new algorithms to compute modular reduction in those kind of representations. We also proposed a new WoP-PBS technique to efficiently evaluate generic LUTs over these new ciphertext types. We used our new optimization framework to provide optimal parameters, for those large homomorphic integers as well as for other practical applications.

The new optimization framework allowed us to convert a crypto-free use-case into an efficient FHE circuit with its associated parameters. It also enables to compare several of the bootstrapping algorithms described in the literature and our new WoP-PBS in diverse scenarios. As a result, we know for each of them, the contexts where they are suited the most. This knowledge will be useful to accelerate an optimization process that has many choices in terms of bootstrapping algorithms.

Future Work. An interesting future work would be to extend the optimization to more than the cryptographic parameters. For instance, many new variables come with the large homomorphic integer representation we introduced in this paper, such as the moduli or the number of blocks, and they could also be optimized instead of being picked by hand.

A second future work consists in optimizing the topology of the graph of FHE operators by offering to the optimization many different options.

Finally, we could use our new optimization framework to find optimal parameters for more FHE schemes, other than the TFHE-like ones.

References

[ACS20] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 345–363. Springer, 2020.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 2015.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. *IACR Cryptology ePrint Archive*, 2012.

[BST20] Florian Bourse, Olivier Sanders, and Jacques Traoré. Improved secure integer comparison via homomorphic encryption. In *CT-RSA*. Springer, 2020.

[CAS17] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. A multi-start heuristic for multiplicative depth minimization of boolean circuits. In Ljiljana Brankovic, Joe Ryan, and William F. Smyth, editors, *Combinatorial Algorithms - 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, July 17-21, 2017, Revised Selected Papers*, volume 10765 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2017.

[CCR19] Hao Chen, Ilaria Chillotti, and Ling Ren. Onion ring ORAM: efficient constant bandwidth oblivious RAM from (leveled) TFHE. In *CCS 2019*. ACM, 2019.

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, 2015.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 2020.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In *CT-RSA*. Springer, 2019.

[CJL+20] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Concrete: Concrete operates on ciphertexts rapidly by extending TfhE. In *WAHC 2020*, 2020.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *CSCML 2021*. Springer, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*, 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 *ASIACRYPT 2021*. Springer, 2021.

[CMG+18] Eduardo Chielle, Oleg Mazonka, Homer Gamil, Nektarios Georgios Tsoutsos, and Michail Maniatakos. E3: A framework for compiling c++ programs with encrypted operands. Cryptology ePrint Archive, Paper 2018/1013, 2018.

[CZB+22] Pierre-Emmanuel Clet, Martin Zuber, Aymen Boudguiga, Renaud Sirdey, and Cédric Gouy-Pailler. Putting up the swiss army knife of homomorphic calculations by means of tfhe functional bootstrapping. Cryptology ePrint Archive, Report 2022/149, 2022. https://ia.cr/2022/149.

[DKS+20] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, jun 2020.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015*, 2015.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[GKT22] Charles Gouert, Rishi Khan, and Nektarios Georgios Tsoutsos. Optimizing homomorphic encryption parameters for arbitrary applications. Cryptology ePrint Archive, Paper 2022/575, 2022. https://eprint.iacr.org/2022/575.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013*. Springer, 2013.

[Kle22] Jakub Klemsa. Hitchhiker's guide to a practical automated TFHE parameter setup for custom applications. *IACR Cryptol. ePrint Arch.*, page 1315, 2022.

[KO22] Jakub Klemsa and Melek Onen. Parallel operations over tfhe-encrypted multi-digit integers. Cryptology ePrint Archive, Report 2022/067, 2022.

[KS21] Kamil Kluczniak and Leonard Schild. FDFB: full domain functional bootstrapping towards practical fully homomorphic encryption. *CoRR*, 2021.

[LLOY20] DongKwon Lee, Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi. Optimizing homomorphic evaluation circuits by program synthesis and term rewriting. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 503–518. ACM, 2020.

[LMP21] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using fhew/tfhe bootstrapping. Cryptology ePrint Archive, Report 2021/1337, 2021. https://ia.cr/2021/1337.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*. Springer, 2010.

[MML+22] Johannes Mono, Chiara Marcolla, Georg Land, Tim Güneysu, and Najwa Aaraj. Finding and evaluating parameters for BGV. *IACR Cryptol. ePrint Arch.*, page 706, 2022.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC 2005*. ACM, 2005.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT 2009*. Springer, 2009.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. *CoRR*, 2021.

Acronyms

$\mathcal{A}^{(\textbf{this work})}$ type of atomic pattern introduced in this paper. 3, 38–41, 43–45, 54–56

$\mathcal{A}^{(\textbf{CGGI20})}$ type of atomic pattern introduced in [CGGI20]. 52–54

$\mathcal{A}^{(\textbf{CJP21})}$ type of atomic pattern introduced in [CJP21]. 3, 24–27, 38–40, 42–45, 49–58

$\mathcal{A}^{(\textbf{GBA21})}$ type of atomic pattern introduced in [GBA21]. 3, 26, 27, 38–40, 47

$\mathcal{A}^{(\textbf{KS-free})}$ type of atomic pattern composed of a dot product and a PBS. 53, 54

$\mathcal{A}^{(\textbf{LMP21})}$ type of atomic pattern introduced in [LMP21]. 3, 40, 41

AP atomic pattern. 22–24, 26, 27, 39, 41–44, 47, 49, 51–54, 56–58

BR blind rotation. 11, 24

BSK bootstrapping key. 12, 57

CRT Chinese Remainder Theorem. 3, 6–9, 15–17, 28, 31, 32, 37, 38, 44–46, 60, 70

DAG directed acyclic graph. 18, 20–23, 48, 49, 55, 57, 59

DP dot product. 23, 24, 38, 40, 41, 49, 51–53, 57, 58

FFT Fast Fourier Transform. 23, 57, 58

FHE fully homomorphic encryption. 3, 5–11, 14, 18–24, 26, 38, 48, 49, 51, 53, 55, 57–60

GCD Greatest Common Divisor. 52

GGSW generalized GSW [GSW13]. 11, 12, 33, 37

GLWE general learning with errors. 10–12, 15, 18–20, 37, 55, 56, 68

KS key switching. 11, 12, 20, 22, 24, 27, 36, 37, 40, 41, 49, 52, 53, 56–58

KSK key switching key. 11, 56, 57

LSB least significant bit(s). 14, 23, 58, 67, 69

LUT lookup table. 3, 5, 7, 8, 11, 12, 14–17, 27–29, 32–34, 37–41, 44, 48, 60, 67–70

LWE learning with errors. 3–5, 9–13, 15–20, 32, 33, 37, 38, 46, 55–57, 60, 66–68, 70

MS modulus switching. 11, 24, 53

MSB most significant bit(s). 13, 14, 30, 32, 37, 58, 67, 69

PBS programmable bootstrapping. 3–5, 7, 11–17, 20, 22–30, 32, 33, 36–40, 45, 46, 48, 49, 51–53, 56, 58, 67, 68, 70

RLWE ring learning with errors. 5, 10, 11

SE sample extraction. 11, 24

WoP-PBS without padding programmable bootstrapping. 3, 7, 17, 28, 32–34, 37–41, 44–46, 49, 60

Supplementary Material

A Details on Advanced Modular Arithmetic from Single LWE Ciphertexts

In this section we provide more details on the advanced modular arithmetic operations from Section 2.2.

A.1 Arithmetic Operators

Thanks to the carry subspace, we can compute leveled operations such as homomorphic additions or multiplications with a known constant. Let's consider two LWE ciphertexts ct_1 and ct_2 encrypting respectively m_1 and m_2 with respective degrees of fullness $\deg_1$ and $\deg_2$. From the degree of fullness one can infer respective worst case messages μ_1 and μ_2 (as defined in 2.2). The following operations are allowed as long as both ciphertexts share the same base β, carry-message modulus p and ciphertext modulus q. Our plaintext format does not allow any native modular reduction modulo β, i.e. the carry subspace will contain the quantity overlapping β and we cannot have a degree of fullness greater than 1 at any time for correctness reason.

Addition To compute the addition $m_1 + m_2$ modulo β one can use the traditional LWE addition. With this approach, the necessary condition to guarantee correctness is $\deg_1 + \deg_2 \leq 1$ and the output ciphertext will have a degree equal to $\deg_1 + \deg_2$.

Multiplication To compute a multiplication between ct_1 (as defined above) and an integer constant $0 \leq c$ one can use the trivial multiplication between an LWE ciphertext and a positive integer. With this approach, the necessary condition to guarantee correctness is $c \cdot \deg_1 \leq 1$ and the output ciphertext will have a degree equal to $c \cdot \deg_1$.

Opposite To compute the opposite of m_1 modulo β we can use the trivial algorithm where we compute the opposite of every elements of the LWE ciphertext ct_1. However, without a correction, this will lead to an encoding of a message that is no more between 0 and $p-1$. This is why one must add the correction term $\beta \cdot \left\lceil \frac{\mu_1}{\beta} \right\rceil$ after computing the opposite of the each coefficients of the ciphertext. With this approach, the necessary condition to guarantee correctness is $\beta \cdot \left\lceil \frac{\mu_1}{\beta} \right\rceil \leq p-1$ and the output degree of fullness is $\beta \cdot \left\lceil \frac{\mu_1}{\beta} \right\rceil \cdot \frac{1}{p-1}$.

Subtraction To compute the subtraction $m_1 - m_2$ modulo β one starts by homomorphically compute the opposite of ct_2 and then compute the homomorphic addition with ct_1. Both the condition and the output degree of fullness can be inferred from the descriptions of the previous operations.

LUT Evaluation The PBS is an operation that allows to evaluate a uni-variate function on the input at the same time as it reduces the noise. It is then easy to compute $l(m_1)$ homomorphically from ct_1 with l a LUT. The requirement is that $\deg_1 \leq 1$, and the output degree is $\frac{\mu_l}{p-1}$ where μ_l is the biggest possible output of the LUT.

A.2 Multiplications

To compute the addition $m_1 \cdot m_2$ we can use a combination of leveled operations and PBS. Here, we propose three types of multiplication:

(i) the multiplication of the two inputs without any modular reduction returning $m_1 \cdot m_2$ (requiring that $\mu_1 \cdot \mu_2 < p$). The output degree of fullness is $\deg_1 \cdot \deg_2 \cdot (p-1)$;

(ii) the multiplication in the LSB, returning an LWE encryption of $m_1 \cdot m_2 \mod \beta$. The output degree of fullness is the minimum between $\frac{\beta-1}{p-1}$ and $\deg_1 \cdot \deg_2 \cdot (p-1)$;

(iii) the multiplication in the MSB, returning an LWE encryption of $\left\lfloor \frac{m_1 \cdot m_2}{\beta} \right\rceil$. The output degree of fullness is $\left\lfloor \frac{\mu_1}{\beta} \right\rceil \cdot \left\lfloor \frac{\mu_2}{\beta} \right\rceil$;

The first method can be used to compute a multiplication of type (ii), and is known in the TFHE literature (see as instance [CJL+20]) and consists in computing the multiplication by observing that $x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$ modulo β. Then, we compute $m_1 + m_2$ and $m_1 - m_2$ in a leveled fashion, and we use two KS-PBS with the LUT computing the uni-variate function $\left\lfloor \frac{x^2}{4} \right\rceil \mod \beta$ to compute $\frac{(m_1+m_2)^2}{4} \mod \beta$ and $\frac{(m_1-m_2)^2}{4} \mod \beta$. We finally subtract the two results and perform another KS-PBS with LUT computing $x \mod \beta$ to get the right result.

The second method we propose is using the Chained-PBS. It can be used to compute multiplications of type (i), type (ii) and type (iii), and requires the use of the technique presented in one of the previous paragraphs and illustrated in Figure 3.

We evaluate the multiplication of type (ii), as a bi-variate function, by shifting one of the two ciphertexts, adding to the other one and by performing a KS-PBS with LUT computing the function $(x \mod (\mu_1+1)) \cdot \left(\left\lfloor \frac{x}{\mu_1+1} \right\rceil \mod \beta \right) \mod \beta$. We evaluate the multiplication of type (iii) in the same manner. The only difference is that the KS-PBS evaluates a LUT computing the function $\left\lfloor \frac{(x \mod (\mu_1+1)) \cdot \left(\left\lfloor \frac{x}{\mu_1+1} \right\rceil \mod \beta \right)}{\beta} \right\rceil$. We evaluate the multiplication of type (i) in the

same manner. The only difference is that the KS-PBS evaluates a LUT computing the function $(x \mod (\mu_1 + 1)) \cdot \left\lfloor \frac{x}{\mu_1+1} \right\rfloor$.

The third method can be use to compute the multiplication of type (i). It consists on using a BFV GLWE multiplication as introduced in [CLOT21] for the TFHE context.

A.3 Carry & Message Extractions

As we observed in previous sections, after performing homomorphic operations the degree of fullness increases and the carry subspace might need to be emptied. To do so, we propose two operations: the *carry extract* operation, that allows to extract the carry $\left\lfloor \frac{m_1}{\beta} \right\rfloor$ of m_1 overlapping β into a new LWE ciphertext (one or more), and *message extract* operation, which allows to extract $m_1 \mod \beta$ into a new LWE ciphertext.

Both operations use a PBS (along with key switching in order to come back to the original secret key, if necessary), taking as input the same LWE ciphertext and the same public material (i.e., the bootstrapping and key switching keys), but different LUT. The carry extract uses $P_{\mathsf{carry\text{-}ext}}$: a r-redundant LUT for $x \to \left\lfloor \frac{x}{\beta} \right\rfloor$. The output degree of fullness is $\left\lfloor \frac{\mu_1}{\beta} \right\rfloor$. The message extract uses $P_{\mathsf{msg\text{-}ext}}$: a r-redundant LUT for $x \to x \mod \beta$. The output degree of fullness is the minimum between $\frac{\beta-1}{p-1}$ and $\deg_1$. If the carry subspace is larger than the message subspace, more than one PBS might be required to empty it all along with slightly different LUTs.

B Example on Radix-Based Integers

In this section we give more detail about the homomorphic large integers introduced in this paper.

Multiplication Let's use a toy example to describe how a multiplication between two encrypted radix-based modular integers. We will use $\kappa = 2$, $\beta_0 = 3$, $\beta_1 = 5$ and $\Omega = 15$. We will multiply $\mathsf{msg} = \mathsf{msg}_1 \cdot \mathsf{msg}_2$ modulo 15 with $\mathsf{msg}_1 = 10 = 3 \cdot 3 + 1$, $\mathsf{msg}_2 = 5 = 1 \cdot 3 + 2$ and $\mathsf{msg} = m_1 \cdot 3 + m_0$. What we do in clear is $m_0 = 1 \cdot 2 = 2$ and $m_1 = 3 \cdot 2 + 3 \cdot 1 \cdot 3 + 1 \cdot 1 = 16 = 1$ because it lives modulo $\beta_1 = 5$.

We now need to compute this homomorphically between two radix-based modular integers. We will set $p = 32$. We have four (two for each integers) ciphertexts encrypting modular integers: $\mathsf{ct}_1^{(1)}$ encrypting 3 under (β_1, p) with degree 4/31, $\mathsf{ct}_0^{(1)}$ encrypting 1 under (β_0, p) with degree 2/31, $\mathsf{ct}_1^{(2)}$ encrypting 1 under (β_1, p) with degree 4/31, $\mathsf{ct}_0^{(2)}$ encrypting 2 under (β_0, p) with degree 2/31.

We want to produce a radix-based modular integer that will be composed of two ciphertexts encrypting modular integers: $\mathsf{ct}_1^{(\mathsf{out})}$ under (β_1, p) and $\mathsf{ct}_0^{(\mathsf{out})}$ under

(β_0, p). The computation will be the following: $\mathsf{ct}_0^{(\mathsf{out})} \leftarrow \mathsf{Mul}\left(\mathsf{ct}_0^{(1)}, \mathsf{ct}_0^{(2)}\right)$ and:

$$\mathsf{ct}_1^{(\mathsf{out})} \leftarrow \mathsf{Mul}\left(\mathsf{ct}_0^{(1)}, \mathsf{ct}_1^{(2)}\right) + \mathsf{Mul}\left(\mathsf{ct}_1^{(1)}, \mathsf{ct}_0^{(2)}\right) + \mathsf{LUT\text{-}Eval}\left(x \mapsto \beta_0 \cdot x \mod \beta_1, \mathsf{Mul}\left(\mathsf{ct}_1^{(1)}, \mathsf{ct}_1^{(2)}\right)\right)$$

Note that Mul refers to the multiplication of type (i), and that LUT-Eval refers to the LUT evaluation in the same section and its first input is a description of the LUT to evaluate. The degree of fullness of $\mathsf{ct}_0^{(\mathsf{out})}$ is $4/32$, and the degree of $\mathsf{ct}_1^{(\mathsf{out})}$ is $8/32 + 8/32 + 4/32 = 20/32$. They will encrypt respectively $1 \cdot 2 = 2$ and $1 \cdot 1 + 3 \cdot 2 + (3 \cdot 1 \cdot 3 \mod 5) = 11$ which when decoding will lead to the expected value.

We could definitely have computed the same functionality, i.e. $\mathsf{msg}_1 \cdot \mathsf{msg}_2$, from the other two types of multiplication, and it would have only changed a few details in the algorithm. Also note that we computed LUT-Eval instead of using a simple multiplication with a constant so the output degree does not become too big. Here again there are many different combination of parameters and algorithms that would have produced the desired result. We provide details on this technique in Algorithm 5 in Supplementary Material C.

Opposite For the negation, we can use msg_1 defined above. We start by computing the opposite of the MSB block, i.e. $\mathsf{ct}_1^{(1)}$, so we end up with the message $5 - 3 = 2$. Then we need to compute the *opposite* of the LSB block, i.e. $\mathsf{ct}_1^{(1)}$, so we end up with the message $3 - 1 = 2$, and finally compensate this LSB opposite by subtracting 1 to the MSB block. It means that we will end up with $2 + (5 - 1) = 6$ encrypted in the MSB block. After decoding, it will output the expected value.

LUT Evaluation There are many examples of univariate functions (such as the inverse). These functions could be computed with the techniques we just described and then, by using a similar method than the one we used to compute the multiplication, it is also possible to compute a homomorphic division of the form $\left\lfloor \frac{m_1}{m_2} \right\rceil$.

C Detail about Algorithms

In this section we provide algorithms that are used in the main body of this paper.

Let $\vec{\beta} = (\beta_0, \ldots, \beta_{\kappa-1})$ and $\vec{p} = (p_0, \ldots, p_{\kappa-1})$. We need to define recursively the quantities $q_{i,\vec{\beta}}$, $r_{i,\vec{\beta}}$ and $\gamma_{\vec{\beta}}$, for $i \in \mathbb{Z}_{\geq -1}$, which are useful in these algorithms, as:

- $q_{i,\vec{\beta}}(x) = \begin{cases} x, \text{ if } i = -1 \\ \left\lfloor \frac{q_{i-1,\vec{\beta}}(x)}{\beta_i} \right\rceil, \text{ if } i \geq 0 \end{cases}$
- $r_{i,\vec{\beta}}(x) = q_{i-1,\vec{\beta}}(x) - q_{i,\vec{\beta}}(x) \cdot \beta_i, \ i \geq 0$
- $\gamma_{\vec{\beta}}(x) = \begin{cases} \min(i \in \Omega), \ \Omega = \{i < |\vec{\beta}|, \ q_{i,\vec{\beta}}(x) = 0\} \\ |\vec{\beta}| \text{ if } \Omega = \varnothing. \end{cases}$

The first algorithm we need is the Decomposition algorithm: we give details in Algorithm 4.

The second tool we need is a padding algorithm, which allows to change the size of a radix-based large integer encryption from κ to κ_{out}. To do so, we complete the ciphertext with trivial encryptions of zero. This is useful when we use the Decomp Algorithm 4 since the output ciphertext might not be of length κ.

The signature of the algorithm is:

$$\boxed{\left(c'_0, \ldots, c'_{\kappa_{\mathsf{out}}-1}\right) \leftarrow \mathsf{Pad}\left(\left(c_0, \ldots, c_{\kappa-1}\right), \alpha, \vec{p}_{\mathsf{out}}, \vec{\beta}_{\mathsf{out}}\right)}$$

We give details on the multiplication operation for radix-based modular integers in Algorithm 5.

C.1 Tree PBS approach on Radix-Based Modular Integers

In this section, we give more details on how to apply the TreePBS technique by [GBA21] to our new radix-based modular integers.

In [GBA21] the plaintext integers are all encrypted under the same basis β: we offer here the possibility to evaluate a large look-up table with integers set in different basis $(\beta_0, \ldots, \beta_{\kappa-1})$.

Let $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$, and let $L = [l_0, l_1, \cdots, l_{\Omega-1}]$ be a LUT with Ω elements. We want to evaluate this LUT on a radix-based modular integer encrypting a message $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \prod_{j=0}^{i-1} \beta_j$.

Then, to evaluate the new multi-radix tree-PBS we performs the following steps:

1. We note as $B = \{\beta_i | i \in [\![0, \kappa-1]\!]\}$ and as $\vartheta(\beta_i)$ the component m_i of msg associated to β_i.
2. We define $\beta_{\max} = \max(\beta \in B)$.
3. We split the LUT L into $\nu = \frac{\prod_{\beta_i \in B} \beta_i}{\beta_{\max}}$ smaller LUTs $(L_0, \ldots, L_{\nu-1})$ that each contain $\beta_{\max}$ different elements of L.
4. We compute a PBS on each of the ν LUTs using the ciphertext encrypting $\vartheta(\beta_{\max})$ as a selector.
5. We build a new large look-up table L by packing, with a key switching, the results of the ν iterations of the PBS in previous step.
6. We remove $\beta_{\max}$ from B: $B = B - \beta_{max}$.
7. We repeat the steps from 2 to 6 until B is empty.

The generalized multi-radix tree-PBS takes as input a radix-based modular integer ciphertext, a large look-up table L and the public material required for the PBS and key switching and returns a LWE ciphertext. The signature is: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Tree\text{-}PBS}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB}, L)$.

For the CRT-only and hybrid approaches, the multi-radix tree-PBS works in the same way.

Algorithm 2: $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{ModReduction}_2((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB})$

Context: $\begin{cases} \vec{\nu} = (\nu_0, \nu_1, \ldots, \nu_{\kappa-1}) \text{ be a convenient decomposition s.t.} \\ \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = \nu_0 + \nu_1\beta_0 + \nu_2\beta_0\beta_1 + \ldots + \nu_{\kappa-2}\prod_{j=0}^{\kappa-3}\beta_j \end{cases}$

Input: $\begin{cases} (\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \text{ encrypting } \mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \left(\prod_{j=0}^{i-1}\beta_i\right) \\ \text{s.t. } \mathsf{ct}_i \text{ encrypts message } m_i \text{ with parameters } (\beta_i, p_i) \end{cases}$

Output: $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right)$ encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \left(\prod_{j=0}^{i-1}\beta_i\right) \mod \Omega$

`/* Copy input and set the` $\kappa - 1$ `block to zero (trivial encryption) */`

$\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow (\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-2}, 0)$

for $j \in [\![0; \kappa-2]\!]$ **do**

  `/* Multiply block` $\kappa - 1$ `times` ν_j`, Multiplication with a Positive Constant as in Section 2.2 */`

  if $\nu_j < 0$ **then**

    $c_j \leftarrow \mathsf{ScalarMul}(\mathsf{ct}_{\kappa-1}, -\nu_j)$

  else

    $c_j \leftarrow \mathsf{ScalarMul}(\mathsf{ct}_{\kappa-1}, \nu_j)$

  `/* Decompose (as in Supplementary Material C)` c_j `block starting from the` β_j `*/`

  $(c_{j,0}, \ldots, c_{j,\kappa-j-1}) \leftarrow \mathsf{Decomp}\left(c_j, (\beta_i)_{i\in[\![j,\kappa-1]\!]}, (p_i)_{i\in[\![j,\kappa-1]\!]}, \mathsf{PUB}\right)$

  `/* Pad (as in Supplementary Material C) the carry to fit with the output */`

  $\left(c'_{j,0}, \ldots, c'_{j,\kappa-1}\right) \leftarrow \mathsf{Pad}\left((c_{j,0}, \ldots, c_{j,\kappa-j-1}), j, (\beta_i)_{i\in[\![0,\kappa-1]\!]}, (p_i)_{i\in[\![0,\kappa-1]\!]}\right)$

  `/* Update the output */`

  if $\nu_j < 0$ **then**

    $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{Add}\left(\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right), \left(c'_{j,0}, \ldots, c'_{j,k-1}\right)\right)$

  else

    $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{Sub}\left(\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right), \left(c'_{j,0}, \ldots, c'_{j,k-1}\right)\right)$

return $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right)$

Algorithm 3: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{WoP\text{-}PBS}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB}, L)$

Context:
- Δ_i : scaling factor for the ciphertext ct_i
- δ_i : bits occupied by message in ciphertext ct_i starting from Δ_i
- $\Omega = 2^{\sum_{i=0}^{\kappa-1} \delta_i}$
- $(\beta_{\mathbf{CB}}, \ell_{\mathbf{CB}})$: the base and level of the output GGSW ciphertexts to the circuit bootstrapping
- $(\varkappa, \vartheta) \in \mathbb{N} \times \mathbb{N}$ defining the modulus switching in the generalized PBS [CLOT21]

Input:
- $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1})$ encrypting $\mathsf{msg} = (m_0, \ldots, m_{\kappa-1})$ with for all $0 \leq i < \kappa$, $\mathsf{Decode}(\mathsf{Decrypt}(\mathsf{ct}_i)) = m_i$
- PUB : public keys required for the whole algorithm
- $L = [l_0, l_1, \cdots, l_{\Omega-1}]$: a LUT, s.t. $l_h \in \mathbb{Z}_\omega$

Output: $\mathsf{ct}_{\mathsf{out}}$ encrypting l_{msg}

for $i \in [\![0; \kappa - 1]\!]$ **do**
  for $j \in [\![0; \delta_i - 2]\!]$ **do**
    `/* Extract from the LSB of the message (use generalized PBS from [CLOT21]) */`
    $\epsilon_i = \frac{q}{4}$
    if $j == 0$ **and** $\frac{q}{p_i} \notin \mathbb{N}$ **then**
      `/* Case of the native CRT (see proof below) */`
      $\epsilon_i \leftarrow \left\lfloor \frac{q \cdot 2^{k-2}}{\beta_i} \right\rceil$
    $\alpha_{i,j} = \frac{\Delta_i \cdot 2^j}{2}$
    $L_{i,j} = [-\alpha_{i,j}, \cdots, -\alpha_{i,j}]$
    $c_i \leftarrow \mathsf{KS\text{-}PBS}\left(\left(\mathsf{ct}_i \cdot 2^{\delta_i - 1 - i}\right) + (0, \cdots, 0, \epsilon_i), \mathsf{PUB}, L_{i,j}, (\varkappa = \log_2(\Delta_i) + j, \vartheta = 0)\right)$
    $c'_i \leftarrow c_i + (0, \cdots, 0, \alpha_{i,j})$
    `/* Subtract the extracted bit from the original ciphertext */`
    $\mathsf{ct}_i \leftarrow \mathsf{Sub}(\mathsf{ct}_i, c'_i)$
    `/* Circuit bootstrap [CGGI20] the extracted bit into a GGSW */`
    $\overline{\overline{C}}_{i,j} \leftarrow \mathsf{CircuitBootstrap}(c'_i, \mathsf{PUB}, (\beta_{\mathbf{CB}}, \ell_{\mathbf{CB}}), (\varkappa = \log_2(\Delta_i) + j, \vartheta = 0))$
  `/* Circuit bootstrap [CGGI20] the last bit into a GGSW */`
  $\overline{\overline{C}}_{i,j} \leftarrow \mathsf{CircuitBootstrap}(\mathsf{ct}_i, \mathsf{PUB}, (\beta_{\mathbf{CB}}, \ell_{\mathbf{CB}}), (\varkappa = \log_2(\Delta_i) + \delta_i - 1, \vartheta = 0))$
`/* Vertical Packing LUT evaluation [CGGI20] */`
$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{VPLut}\left(\left\{\overline{\overline{C}}_{i,j}\right\}_{i \in [\![0;\kappa-1]\!]}^{j \in \lceil 0; \delta_i - 1 \rceil}, L\right)$
return $\mathsf{ct}_{\mathsf{out}}$

Algorithm 4: $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]} \leftarrow \mathsf{Decomp}\left(\mathsf{ct}_{\mathsf{in}}, \vec{\beta}, \vec{p}, \mathsf{PUB}\right)$

Context:
$$\begin{cases} (q, p, \deg): \text{ parameters of } \mathsf{ct}_{\mathsf{in}} \\ \mu := \deg \cdot (p-1) \\ \gamma := \gamma_{\vec{\beta}}(\mu) \\ \vec{s} \in \mathbb{Z}^n : \text{the secret key} \\ P_{i,\vec{\beta}} : \text{a LUT for } x \to r_{i,\vec{\beta}}(x) \cdot \frac{q}{2\cdot p_i}, i \in [\![0, \kappa-1]\!] \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_{\mathsf{in}} : \text{ LWE encryption of a message } m \\ (\vec{p}, \vec{\beta}) \in \mathbb{N}^{\kappa^2} \\ \mathsf{PUB}: \text{ public material for KS-PBS} \end{cases}$$

Output: $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]}$ encrypting the message m

for $j \in [\![0, \gamma]\!]$ **do**
  $\mathsf{ct}_j \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{PUB}, P_i)$
  with ct_j LWE encryption with parameters $\left(q, \beta_j, p_j, \deg = \min(\frac{\beta_j - 1}{p_j - 1}, \frac{q_{j-1,\vec{\beta}}(\mu)}{p_j - 1})\right)$
end
return $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]}$

Algorithm 5: $\left(\mathsf{ct}_0^{(\mathsf{out})}, \ldots, \mathsf{ct}_{\kappa-1}^{(\mathsf{out})}\right) \leftarrow \mathsf{Mult}\left(\left(\mathsf{ct}_0^{(1)}, \ldots, \mathsf{ct}_{\kappa_1-1}^{(1)}\right), \left(\mathsf{ct}_0^{(2)}, \ldots, \mathsf{ct}_{\kappa_2-1}^{(2)}\right), \mathsf{PUB}\right)$

Context:
$$\begin{cases} (q, p_{j,i}, \beta_{j,i}, \deg_{j,i}): \text{ paremathers of } \mathsf{ct}_i^{(j)} \\ \mu_{j,i} := \deg_{j,i} \cdot (p_{j,i} - 1) + 1, \ j \in [\![1,2]\!], \ i \in [\![0, \kappa_j - 1]\!] \\ \vec{\beta}_j := (\beta_{j,0}, \ldots, \beta_{j,\kappa_j-1}), \ \vec{p}_j := (p_{j,0}, \ldots, p_{j,\kappa_j-1}) \\ \vec{\beta}_{j,i} := (\beta_{j,i}, \ldots, \beta_{j,\kappa_j-1}), \ \vec{p}_{j,i} := (p_{j,i}, \ldots, p_{j,\kappa_j-1}) \\ \gamma_{h,k} := \gamma_{\overrightarrow{\beta}_{2,k}}((\mu_{1,h}-1)\cdot(\mu_{2,k}-1)) \\ \{P_{i,r_k,\overrightarrow{\beta_{2,j}}}\}_{0\le i\le\kappa_1-1,\ 0\le j\le\kappa_2-1,\ 0\le k\le\gamma_{i,j}} : \text{a LUT for} \\ x \to r_{k,\overrightarrow{\beta}_{2,j}}\left((x \mod \mu_{2,i}) \cdot \left\lfloor \frac{x}{\mu_{2,i}} \right\rfloor\right) \cdot \frac{q}{2\cdot p_{2,k+j}} \end{cases}$$

Input:
$$\begin{cases} \left(\mathsf{ct}_0^{(i)}, \ldots, \mathsf{ct}_{\kappa_i-1}^{(i)}\right) : \text{ encrypting } \mathsf{msg}_i \text{ under } \vec{s}, \ i \in [\![1,2]\!] \\ \mathsf{PUB}: \text{ public material for KS-PBS} \end{cases}$$

Output: $\left(\mathsf{ct}_0^{(\mathsf{out})}, \ldots, \mathsf{ct}_{\kappa-1}^{(\mathsf{out})}\right)$ encrypting $\mathsf{msg}_1 \cdot \mathsf{msg}_2$ under $\vec{s}$

for $i \in [\![0; \kappa_1 - 1]\!]$ **do**
  `/* Put the block to the right basis */`
  $\mathsf{ct}_{tmp} \leftarrow \mathsf{LweBasisChange}(\mathsf{ct}^{(1)}i, p_{2,0}, \beta_{2,0}, \mathsf{PUB})$
  `/* Compute the multiplication-decomposition */`
  $\left(\mathsf{ct}_0^{(tmp)}, \ldots, \mathsf{ct}_{\kappa-1}^{(tmp)}\right) \leftarrow \mathsf{OneBlockMul}\left(\mathsf{ct}_{tmp}, \left(\mathsf{ct}_0^{(2)}, \ldots, \mathsf{ct}_{\kappa_2-1}^{(2)}\right), \mathsf{PUB}\right)$
  `/* Add the results of the multiplications together */`
  $\left(\mathsf{ct}_0^{(\mathsf{out})}, \ldots, \mathsf{ct}_{\kappa-1}^{(\mathsf{out})}\right) \leftarrow \mathsf{Add}(\left(\mathsf{ct}_0^{(\mathsf{out})}, \ldots, \mathsf{ct}_{\kappa-1}^{(\mathsf{out})}\right), \left(\mathsf{ct}_0^{(tmp)}, \ldots, \mathsf{ct}_{\kappa-1}^{(tmp)}\right))$
end
return $\left(\mathsf{ct}_0^{(\mathsf{out})}, \ldots, \mathsf{ct}_{\kappa-1}^{(\mathsf{out})}\right)$