TFHE Gets Real: an Efficient and Flexible Homomorphic Floating-Point Arithmetic

Loris Bergerat[1,2], Ilaria Chillotti* , Damien Ligier* , Jean-Baptiste Orfila[1], and Samuel Tap[1]

[1]Zama, Paris, France - https://zama.ai/
{loris.bergerat, jb.orfila, samuel.tap}@zama.ai
[2]Université Caen Normandie, ENSICAEN, CNRS, Normandie Univ, GREYC UMR 6072, F-14000 Caen, France

Abstract

Floating-point arithmetic plays a central role in computer science and is used in various domains where precision and computational scale are essential. One notable application is in machine learning, where Fully Homomorphic Encryption (FHE) can play a crucial role in safeguarding user privacy. In this paper, we focus on TFHE and develop novel homomorphic operators designed to enable the construction of precise and adaptable homomorphic floating-point operations. Integrating floating-point arithmetic within the context of FHE is particularly challenging due to constraints such as small message space and the lack of information during computation. Despite these challenges, we were able to determine parameters for common precisions (e.g., 32-bit, 64-bit) and achieve remarkable computational speeds, with 32-bit floating-point additions completing in 2.5 seconds and multiplications in approximately 1 second in a multi-threaded environment. These metrics provide empirical evidence of the efficiency and practicality of our proposed methods, which significantly outperform previous efforts. Our results demonstrate a significant advancement in the practical application of FHE, making it more viable for real-world scenarios and bridging the gap between theoretical encryption techniques and practical usability.

*Ilaria Chillotti and Damien Ligier contributed to this work during a previous employment at Zama.

Contents

1 Introduction

Fully homomorphic encryption (FHE) is a cryptographic paradigm that allows evaluating any circuit over encrypted data, ensuring that the decrypted output contains the results of the circuit evaluation. The concept of FHE was first proven possible in the seminal paper by Gentry [Gen09]. Since then, substantial work has been done to enhance performance. As of today, the following schemes are considered to be practical, BGV [BGV12], BFV [Bra12, FV12], CKKS [CKKS17, CHK+18], FHEW [DM15], TFHE [CGGI20]. All these schemes are based on the Learning With Errors (LWE) problem [Reg05], its Ring variants (RLWE) [SSTX09, LPR10] and the General LWE (GLWE), also called Module LWE [LS15, BGV12]. However, there is a variation in their approach during computations: while BGV/BFV/FHEW/TFHE perform exact integer computations, CKKS generally focuses on approximate computations using fixed-point arithmetic. The easiest solution to adapt the other schemes to support real numbers is to define a special encoding mapping modular integers to fixed-point numbers as done in [CJP21, CJL+20] for TFHE, or in [CSVW16] for BGV and [Lai17] for BFV. A significant limitation of this method, inherent to fixed-point arithmetic, is that the encoding range must be sufficiently large to cover all values used in the application. Otherwise, the cumulative precision loss during computations becomes excessively large. In the case of TFHE, this problem is twofold: one must either extend the precision and/or bound the circuit depth. The former significantly impacts performance, as the bootstrapping efficiency is closely linked to the number of bits required to represent a message. In practical terms, TFHE's bootstrapping is deemed efficient when the precision is smaller than 10 bits [BBB+22]. The latter is a direct contradiction with the TFHE construction, whose main advantage is to have the possibility to evaluate any circuit depths through the functional/programmable bootstrapping. Therefore, the only feasible approach to handle real numbers within TFHE is to employ floating-point representation. Thus, the problem we are solving in this paper is: *How to design an efficient and customizable floating-point arithmetic based on TFHE?*

1.1 Our Results

In the current landscape, FHE schemes, and especially TFHE, are not designed to seamlessly integrate with established floating-point standards,

which are tailored to match hardware specifications, such as bit sizes and the frequent use of conditional instructions. This paper introduces novel homomorphic operators that can serve as the building blocks for constructing homomorphic floating-point arithmetic with arbitrary precision within a given FHE framework. As a matter of fact, our method is compliant with any FHE scheme providing a bootstrapping which is exact and programmable, so FHEW/TFHE and any of their more optimized versions ([BIP+22, LMK+23]) should work.

Our innovative operators not only align with widely recognized floating-point arithmetic standards like IEEE754 [Ins08], but also adapt to variants with custom precision levels, boosting performance. This marks the first feasible approach for performing homomorphic computation with floating-points, significantly surpassing previous methods in time efficiency by a factor of at least 10.

Currently, there is a substantial gap between cleartext floating-point arithmetic and homomorphic floating-point arithmetic. Even if there remains a considerable distance to achieving a fully practical deployment, our work consistently narrows this gap. While not yet practical for general-purpose applications, our solution may already be sufficiently efficient for certain specialized use cases.

In TFHE, the process to encrypt large integers involves dividing the integer message into several parts, each of which is then encrypted as its own LWE ciphertext. This method is crucial for handling large numerical values effectively by encrypting them in smaller, discrete units [BST20, GBA21, KO22, CZB+22, LMP21, CLOT21, BBB+22]. Then, the operations are either modular, or the number of blocks increases all along the computation. To ensure correctness, part of the cleartext space, referred to as the carry space, is reserved to accommodate the growth of the message during computation. Each ciphertext is also paired with public metadata, referred to as the *degree* in previous studies [BBB+22], which tracks the maximum potential size of the message. When the public metadata indicates that the carry space is nearing capacity, the carry buffer is either cleared or a new block is added at each operation. This approach is based on worst-case size estimations, meaning that in many cases, additional operations could be performed before needing to clear the carry space or add a new block.

To construct floating-point arithmetic, we must represent mantissas and exponents, which can be viewed as large integers. We aim to make this arithmetic as efficient as possible, which requires circumventing the limitations of

worst-case estimations. In fact, an inherent challenge of FHE is the lack of information about underlying values, due to the data being encrypted. This limitation prevents the typical optimizations commonly used in floating-point arithmetic implementations, such as those described in [PFH+20].

We introduce novel algorithms that automatically retain only the most significant bits and discard some of the least significant bits, maintaining the same representation throughout. This mimics a well-known round mode of floats, called the rounding towards zero mode. We refer to [MBDD+18, Section 2.2.1] or to [Hwa24] for more details. Our first method, based on [BBB+22], ensures the correct encoding after each operation. This method is effective for small mantissas and exponents but does not scale well. The second method leverages circuit bootstrapping to obtain the carry value, allowing the ciphertext to be updated accordingly. This approach is faster than the first method for large mantissas or exponents.

Building on large integers, we introduce two different methods to build efficient floating-point arithmetic, each with its own pros and cons. The first approach, detailed in Section 3.1, heavily utilizes the alternative PBS proposed in [BBB+22] (referred to as WoP-PBS) to perform operations on floating-point numbers. This method allows for the efficient evaluation of functions on ciphertexts that encrypt large integers. Our approach, while straightforward, distinguishes itself from other homomorphic floating-point methods by representing a floating-point number within a single ciphertext (or more). This technique proves particularly efficient for floating-point numbers with small precision (up to 12 bits).

The second approach, which is the core of this contribution, constructs floating-point arithmetic based on TFHE that can follow standards such as [Ins08] and is not constrained by precision. Our new algorithms make extensive use of an extended version of CMux, a homomorphic operator that selects between two inputs based on an encrypted decision bit. This method is key to developing faster homomorphic operations that effectively combine the sign, mantissa, and exponent of one or more homomorphic floats. During homomorphic float operations, to compute the resulting mantissa, an extra LWE ciphertext will be used to make the operation both faster and more precise. This approach, soberly titled homomorphic floating-point (HFP), is the first that can be considered to be deployed for real use-cases. Practically, on a typical server machine, an addition of two 32-bit HFP numbers takes around 2 seconds, while a multiplication takes around 1 second. Beyond

the description of all arithmetic operations, including the division, we also provide their noise analysis and the hypotheses done to generate the cryptographic parameters and their associated failure probability. We also include algorithms to efficiently compute the ReLU and the sigmoid, two of the most used functions in machine learning. We show how to easily extend our approach to take into account some floating-point subtleties, like the special values. As a simple application and showcase of the versatility of floating-points, we briefly detail how to compute the approximation of any functions. In the end, our method outperforms the state-of-the-art (as shown in the next section), which was more about showing that floating-points might be doable with TFHE, rather than giving a practical solution as we do here.

1.1.1 Prior Approaches

Efforts to develop efficient FHE computation methods for real numbers can be categorized into two approaches: the fixed-point arithmetic approach and the floating-point arithmetic approach. Most of the first attempts [CSVW16, Lai17, AN16] focus on the BFV scheme [Bra12, FV12]. In [CSVW16, Lai17], the authors chose the fixed-point approach. Roughly, their idea is to decompose a real number into two integers, one representing the value before the point and the second representing the value after the point. The binary decomposition of the two integers is encrypted in one RLWE ciphertext such that the integer part is encrypted over the coefficient of small degree and the fractional part is encrypted on the coefficient of high degree. This method encounters two significant limitations: first, after several operations, accuracy is compromised due to the mixing of the fractional and the integer parts of the number. Second, the computation must remain within a certain modulus limit; exceeding this threshold also results in a loss of correctness. Thus, using fixed-point arithmetic is particularly suitable for FHE schemes where the depth of the circuit is somewhat bounded, since they share a similar constraint. In fact, an encrypted floating-point number is often represented by one ciphertext for the sign, one or several ciphertexts for the mantissa and one or several ciphertexts for the exponent. Their approach is also based on FV, whose bootstrapping is neither efficient nor programmable. This results in unpractical methods that cannot be adapted to TFHE.

In the TFHE context, to the best of our knowledge, only two techniques have been studied [ML20] and [LS22]. The former [ML20] uses the tradi-

tional floating-point representation, where each LWE ciphertext contains one boolean value. Then, they rely only on boolean gate operations to perform the floating-point computation. Beyond the lack of space efficiency, the major problem with this solution is its time complexity. In the latter [LS22], the authors take advantage of RLWE ciphertexts to represent floats. They have a floating-point in three parts: an RLWE for the sign, one for the mantissa and another one for the exponent. In this representation, the sign and the exponent are each represented on the first coefficient of their RLWE. The mantissa is represented by several coefficients depending on the base of the decomposition. In this work, they propose a method to detect overflow based on an encrypted witness. As previously, the main drawback lies in the time complexity, which suffers from an exponential factor related to the size of the exponent and the use of tensor product and relinearisation.

In our work, we use the standard representation proposed before (several LWE ciphertexts for the mantissa, several LWE ciphertexts for the exponent and one LWE ciphertext for the sign). The main change in our algorithm is the use of circuit bootstrapping (CBS) which is costlier than a PBS ($\mathbb{C}_{\mathsf{CBS}} \approx \ell_{\mathsf{CBS}} \cdot \mathbb{C}_{\mathsf{PBS}}$) but gives the possibility to perform CMuxes and reduce the cost of the TFHE algorithm used in the floating-point arithmetic. Note that our work could benefit from recent optimizations done in [WWL+24].

The code for [ML20] is not publicly available, and despite significant effort, we were unable to successfully run the code provided with [LS22]. As a result, the comparison below is based on the timings reported in the respective papers. In [ML20], the experiments were conducted on an Intel i7-6700@3.40 GHz (up to 4 GHz) with 8 threads. Since the computational model (i.e., sequential or parallel) is not specified, we assume these experiments were run sequentially. In the latter paper [LS22], experiments were run on an Intel Xeon Silver 4210@2.40 GHz, with 40 threads. As shown in Table 1, our approach significantly outperforms existing methods, achieving at least an 8-fold improvement (for 32-bit floating-point multiplication) and up to 100-fold improvement (for 64-bit floating-point addition). Sequential timings are:

As explained above, each technique was evaluated on different machines, with some assumptions regarding the nature of the computation (sequential or parallel). To provide a more complete comparison, Table 2 presents a complexity comparison in terms of the number of PBS operations required for the main operations (addition and multiplication) between our work and the state of the art. This comparison highlights the removal of the exponential

Paper (Precision)	**Add**	**Mul**
[ML20] (32-bits)	490s	162s
[LS22] (32-bits)	530s	443s
Ours (32-bits)	7s	20s
[ML20] (64-bits)	1200s	686s
[LS22] (64-bits)	858s	808s
Ours (64-bits)	12s	82s

Table 1: Comparison of addition and multiplication times with the state-of-the-art.

factor in the complexity formulas, further emphasizing the efficiency of our approach.

In this context, the terms $\rho_{\mathfrak{m}} \cdot \ell_{\mathfrak{m}}$ and $\rho_{\mathfrak{e}} \cdot \ell_{\mathfrak{e}}$ represent the number of bits in the mantissa and exponent, respectively. In our method, $\rho_{\mathfrak{m}}$ (and $\rho_{\mathfrak{e}}$) corresponds to the message space in each LWE ciphertext for the mantissa (and exponent), while $\ell_{\mathfrak{m}}$ (and $\ell_{\mathfrak{e}}$) indicates the number of ciphertexts representing the mantissa (and exponent). Considering the complexity of the two previous works based on TFHE, we achieve a significant gain by eliminating the exponential factor for addition (present in both previous techniques) and for multiplication specifically in [LS22]

	Addition Complexity	Multiplication Complexity
[ML20]	$\approx 2^{\rho_{\mathfrak{e}}\ell_{\mathfrak{e}}}(\ell_{\mathfrak{m}}\rho_{\mathfrak{m}} + \ell_{\mathfrak{e}}\rho_{\mathfrak{e}})\log(\ell_{\mathfrak{m}}\rho_{\mathfrak{m}} + \ell_{\mathfrak{e}}\rho_{\mathfrak{e}})\ \mathbb{C}_{\mathsf{PBS}}$	$\approx (\ell_{\mathfrak{m}}\rho_{\mathfrak{m}})^2 + \ell_{\mathfrak{e}}\rho_{\mathfrak{e}}\log(\ell_{\mathfrak{e}}\rho_{\mathfrak{e}})\ \mathbb{C}_{\mathsf{PBS}}$
[LS22]	$> 2^{\rho_{\mathfrak{e}}\ell_{\mathfrak{e}}+1}\ \mathbb{C}_{\mathsf{PBS}}$	$> 2^{\rho_{\mathfrak{e}}\ell_{\mathfrak{e}}+1}\ \mathbb{C}_{\mathsf{PBS}}$
Ours	$(3 \cdot \ell_{\mathfrak{m}} + 6 \cdot \ell_{\mathfrak{e}} + 3)\mathbb{C}_{\mathsf{PBS}} + (\ell_{\mathfrak{m}} + \ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}} + 3)\mathbb{C}_{\mathsf{CBS}}$	$\left(\frac{\ell_{\mathfrak{m}}{}^2}{2}\left(1 + \frac{1}{\rho_{\mathfrak{e}}}\right) + \ell_{\mathfrak{m}}\left(2 + \frac{1}{\rho_{\mathfrak{e}}}\right) + 4\right)\mathbb{C}_{\mathsf{PBS}} + \mathbb{C}_{\mathsf{CBS}}$

Table 2: Our Method vs. Existing Works.

1.1.2 Roadmap

In Section 2, we give an overview of TFHE and floating-point arithmetic. Section 3 presents our initial efforts in developing efficient floating-point arithmetic using WoP-PBS from [BBB+22], followed by a new, more versatile encoding methodology for floating-points tailored for TFHE, along with fundamental building blocks for advanced operations. Sections 4 and 5 focus on detailing algorithms for floating-point number computation. Section 6 highlights the practicality of our methods, demonstrating their efficiency with

timing benchmarks for standard float precision. Finally, Section 7 describes how to improve our representation with the infinity values and how to compute approximate functions.

2 Background and Notations

2.1 Notations

Let q be a positive integer. We note $\mathbb{Z}_q$ the ring $\mathbb{Z}/q\mathbb{Z}$. Let N be a power of two, representing the degree of quotient polynomial. Then, we note $\mathfrak{R}_{q,N}$ the ring $\mathbb{Z}_q[X]/(X^N+1)$. By convention, the vectors are written in bold $\boldsymbol{x}$, polynomials in upper case X, and integers are in lower case x. Moreover, the values associated with floating-points will be written in the following manner: $\mathfrak{m}$ for the mantissa, $\mathfrak{e}$ for the exponent, $\mathfrak{s}$ for the sign, and $\mathfrak{z}$ for any other value. The number of ciphertexts associated with $\mathfrak{z}$ is denoted by $l_{\mathfrak{z}}$, and the number of bits encoding this value by $\rho_{\mathfrak{z}}$. Regarding probability distributions, the uniform distribution over a set S is denoted as $\mathcal{U}(S)$ whereas a Gaussian distribution with a mean set to zero and a variance σ^2 is written $\mathcal{N}_{\sigma^2}$.

2.2 FHE Ciphertext Types

TFHE security is based on the Learning With Errors (LWE) assumption [Reg05], its extension to polynomial rings [SSTX09, LPR10] RLWE and the Generalized approach GLWE [BGV12, LS15].

Definition 1 (GLWE Ciphertexts [BGV12, LS15]) *Let $\boldsymbol{S} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$ be the secret key, with $S_i = \sum_{j=0}^{N-1} s_{i,j} X^j$, where each coefficient $s_{i,j}$ is sampled from a uniform binary, uniform ternary or Gaussian distribution. Let $\boldsymbol{A} = (A_0, \cdots, A_{k-1}) \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})^k$ be the mask and let $E \in \mathfrak{R}_{q,N}$ be the noise, where each coefficient e_i is sampled from a Gaussian distribution $\mathcal{N}_{\sigma^2}$. Let $\Delta \in \mathbb{N}$ be the scaling factor depending on the plaintext space p, such that $\Delta = \frac{q}{2 \cdot p}$. A* GLWE *ciphertext of a plaintext $\Delta \cdot M \in \mathfrak{R}_{q,N}$ under a secret key $\boldsymbol{S} \in \mathfrak{R}_{q,N}^k$ is defined as:*

$$\mathsf{CT} = \left(\boldsymbol{A}, B = \sum_{i=0}^{k-1} A_i \cdot S_i + \Delta \cdot M + E \right) \in \mathsf{GLWE}_{\boldsymbol{S}}(\Delta \cdot M) \subseteq \mathfrak{R}_{q,N}^{k+1}.$$

In what follows, Δ is implicit.

Definition 2 (GLEV ciphertexts [CLOT21]) *For a given decomposition base $\beta \in \mathbb{N}^*$ and a level decomposition $\ell \in \mathbb{N}^*$, a* GLEV *ciphertext of a message $M \in \mathfrak{R}_{q,N}$ under a secret key $\boldsymbol{S} \in \mathfrak{R}_{q,N}^k$ is a ciphertext composed of ℓ* GLWE *ciphertexts encrypting the same message M for different scaling factors (given by the base β and the level ℓ). Let $\mathsf{CT}_i \in \mathsf{GLWE}_{\boldsymbol{S}}\left(\frac{q}{\beta^{i+1}}M\right) \subseteq \mathfrak{R}_{q,N}^{k+1}$ for $i \in [0, \ell)$. Then, a* GLEV *ciphertext is defined as:*

$$\overline{\mathsf{CT}} = (\mathsf{CT}_0, \cdots, \mathsf{CT}_{\ell-1}) \in \mathsf{GLEV}_{\boldsymbol{S}}^{\beta,\ell}(M) \subseteq \mathfrak{R}_{q,N}^{\ell\times(k+1)}.$$

Definition 3 (GGSW ciphertexts [GSW13, CLOT21]) *For a given decomposition base $\beta \in \mathbb{N}^*$ and a level decomposition $\ell \in \mathbb{N}^*$, a* GGSW *ciphertext encrypting a message $M \in \mathfrak{R}_{q,N}$ under a secret key $\boldsymbol{S} \in \mathfrak{R}_{q,N}^k$ is composed of $(k+1)$* GLEV *ciphertexts encrypting the same message M multiplied by elements of the secret key for different scaling factors (given by the base β and the level ℓ). Let $\overline{\mathsf{CT}}_j \in \mathsf{GLEV}_{\boldsymbol{S}}^{\beta,\ell}(-S_j \cdot M) \subseteq \mathfrak{R}_{q,N}^{\ell\times(k+1)}$ for $j \in [0, k)$ and $\overline{\mathsf{CT}}_k \in \mathsf{GLEV}_{\boldsymbol{S}}^{\beta,\ell}(M) \subseteq \mathfrak{R}_{q,N}^{\ell\times(k+1)}$. Then, a* GGSW *ciphertext is defined as:*

$$\overline{\overline{\mathsf{CT}}} = (\overline{\mathsf{CT}}_0, \cdots, \overline{\mathsf{CT}}_k) \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(M) \subseteq \mathfrak{R}_{q,N}^{(k+1)\ell\times(k+1)}.$$

Remark 1 (LWE and RLWE) *A* GLWE *ciphertext with $N = 1$ is a* LWE *ciphertext. In this case, we consider $n = k$ for the* LWE *size, and all the elements of the ciphertext are denoted in lower case (i.e., $\mathsf{ct}, \boldsymbol{s}, \boldsymbol{a}$ and b). A* GLWE *ciphertext with $k = 1$ and $N > 1$ is a* RLWE *ciphertext. This can be extended to* GLEV *and* GGSW *ciphertext as well.*

2.3 FHE Operators

In what follows, we recall some of the TFHE operators. We refer the reader to the associated references for more details. We assume that all the correctness conditions are fulfilled when one of these algorithms is called. More details on these conditions are given in lemmas alongside almost every algorithm introduced in this paper (see Lemma 6).

Extract and Insert LWE samples The sample extract is an algorithm taking as input a GLWE ciphertext in $\mathfrak{R}_{q,N}^{k+1}$ and returning its i^{th} coefficient

as an LWE ciphertext in $\mathbb{Z}_q^{kN+1}$ for i in $[0, N)$. This operation is noiseless and is detailed in [CGGI20].

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{SampleExtract}(\mathsf{CT}_{\mathsf{in}}, i).$$

Usually, only the first coefficient is extracted. In this case, this is simply denoted: $\mathsf{SampleExtract}(\mathsf{CT}_{\mathsf{in}}) = \mathsf{SampleExtract}(\mathsf{CT}_{\mathsf{in}}, 0)$.

Conversely, inserting a sample is an operation that creates a GLWE ciphertext from a LWE ciphertext taken as input. The first coefficient of the GLWE ciphertext encrypts the same plaintext as the LWE ciphertext. All the other coefficients encrypt random values. See [BCL+23, Alg. 9] for more details. The signature of this algorithm is:

$$\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{SampleInsert}(\mathsf{ct}_{\mathsf{in}}).$$

Keyswitch (KS) The keyswitch (e.g., detailed in [CGGI20, CLOT21]) is an homomorphic operation which changes the encryption secret key of a ciphertext using only public material. This operation takes as input an LWE ciphertext encrypting a message m under a secret key $\boldsymbol{s}_{\mathsf{in}} \in \mathbb{Z}_q^{n_{\mathsf{in}}}$ and returns an LWE ciphertext encrypting the same message m under a secret key $\boldsymbol{s}_{\mathsf{out}} \in \mathbb{Z}_q^{n_{\mathsf{out}}}$. To perform this operation, a key-switching-key (KSK) is required. A KSK is an encryption of $\boldsymbol{s}_{\mathsf{in}}$ with redundancy under the secret key $\boldsymbol{s}_{\mathsf{out}}$, i.e., $\mathsf{KSK} = (\mathsf{KSK}_0, \cdots, \mathsf{KSK}_{n_{\mathsf{in}}-1})$ where $\mathsf{KSK}_i \in \mathsf{LEV}^{\beta,\ell}_{\boldsymbol{s}_{\mathsf{out}}}(\boldsymbol{s}_{\mathsf{in},i})$ for $i \in [0, n_{\mathsf{in}})$. The signature is:

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{KS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{KSK}).$$

Programmable Bootstrapping (PBS) Unique among bootstrapping techniques, TFHE's bootstrapping not only reduces the noise in a ciphertext but also has the distinct capability to evaluate any univariate function f represented as a Look-Up Table LUT_f. This distinctive feature of evaluating a function is the reason TFHE's bootstrapping is often referred to as the functional or programmable bootstrapping (PBS) [CGGI20, CJL+20, CJP21]. The PBS is done in three steps: a modulus switch (MS), a blind rotation (BR) and a sample extract (SE). Taking as input the bootstrapping key BSK, the look-up table LUT_f and an encrypted LWE ciphertext of a message m under a secret key $\boldsymbol{s}_{\mathsf{in}}$, the bootstrapping outputs an LWE ciphertext which encrypts the message $f(m)$ under the secret key $\boldsymbol{s}_{\mathsf{out}}$ with a smaller noise.

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{LUT}_f, \mathsf{BSK}).$$

The bootstrapping key is an encryption of each element of the input secret key $\boldsymbol{s}_{\mathsf{in}} = (s_0, \cdots s_{n-1})$ as GGSW ciphertexts encrypted under a key $\boldsymbol{S}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^k$ (note that each element of the input secret key $\boldsymbol{s}_{\mathsf{in}}$ is seen as an element of $\mathfrak{R}_{q,N}$ where s_i corresponds to the constant coefficient, and all the other coefficients are set to zero). So we have $\mathsf{BSK} = (\mathsf{BSK}_0, \cdots, \mathsf{BSK}_{n-1})$ where $\mathsf{BSK}_i \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(s_i)$ for $i \in [0, n)$ and where s_i corresponds to an element of the LWE input secret key $\boldsymbol{s}$. As introduced in [CJP21], to improve the computational time of the PBS, a keyswitch always precedes a PBS. In the following, the two operations are merged together.

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{LUT}_f, \mathsf{KSK}, \mathsf{BSK}).$$

Circuit Bootstrapping (CBS) [CGGI20, Alg. 11] The circuit bootstrapping (CBS) is an operation that takes as input an LWE ciphertext encrypting a binary message $m \in \{0, 1\}$ under a secret key $\boldsymbol{s}_{\mathsf{in}}$ and returns a GGSW ciphertext encrypting the same binary message m under the secret key $\boldsymbol{S}_{\mathsf{out}}$.

$$\overline{\overline{\mathsf{CT}}}_{\mathsf{out}} \leftarrow \mathsf{CBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{KSK}, \mathsf{PFKSK}).$$

BSK is the bootstrapping key as defined above and PFKSK stands for private functional key switch key. A $\mathsf{PFKSK} = [\mathsf{PFKSK}_0, \cdots, \mathsf{PFKSK}_k]$ is a public key encrypting the product of each input and output secret key where $\mathsf{PFKSK}_i = \mathsf{PFKSK}_{\boldsymbol{s}_{\mathsf{in}} \to \boldsymbol{S}_{\mathsf{out}}}(S_{\mathsf{out},i}) = [\mathsf{GLEV}_{\boldsymbol{S}_{\mathsf{out}}}^{\beta,\ell}(-s_{\mathsf{in},0} \cdot S_{\mathsf{out},i}), \cdots, \mathsf{GLEV}_{\boldsymbol{S}_{\mathsf{out}}}^{\beta,\ell}(-s_{\mathsf{in},kN-1} \cdot S_{\mathsf{out},i})]$ for $i \in [0, k_{\mathsf{out}}]$ with $S_{\mathsf{out},k} = -1$. With this public key, we can perform an operation with an LWE ciphertext encrypting a message m under the secret key $\boldsymbol{s}$ and PFKSK_i to obtain a GLWE ciphertext under the secret key $\boldsymbol{S}_{\mathsf{out}}$ of the message $m \cdot S_{\mathsf{out},i}$ The CBS operation is done in two steps. The first one consists in performing several bootstrappings on the input LWE ciphertext to obtain many LWE ciphertexts which encrypt the binary message with different scaling factors depending on the decomposition base β and the level decomposition ℓ of the GGSW ciphertext. This operation transforms the input LWE ciphertext $\mathsf{ct} \in \mathsf{LWE}_{\boldsymbol{s}}(m)$ into a LEV ciphertext $\overline{\mathsf{ct}} \in \mathsf{LEV}_{\boldsymbol{s}}^{\beta,\ell}(m)$. The second step consists in using the PFKSK to multiply each of the LWE ciphertext of the LEV obtained after the first operation to get $k+1$ new GLEV ciphertexts $\overline{\mathsf{CT}}$ encrypting $m \cdot S_{\mathsf{out},i}$ for $i \in [0, k]$ (with $S_{\mathsf{out},k} = -1$). This collection of $\overline{\mathsf{CT}}$ corresponds to a GGSW ciphertext as defined in Definition 3.

Remark 2 *In what follows, we constrain the input* LWE *secret key* $\boldsymbol{s}_{\mathsf{in}} = (s_0, \cdots, s_{kN-1})$ *to contain the same coefficients as the flattened* GGSW *output*

secret key $\boldsymbol{S}_{\mathsf{out}} = (S_0, \cdots, S_{k-1})$ *where* $S_i = \sum_{j=0}^{N-1} s_{iN+j} X^j$. *Thanks to this constraint, we can use the sample insert operation to cast an* LWE *ciphertext encrypted under* $\boldsymbol{s}_{\mathsf{in}}$ *as a* GLWE *ciphertext encrypted under* $\boldsymbol{S}_{\mathsf{out}}$.

CMux The CMux selects one of two GLWE ciphertexts, depending on the value of an encrypted decision bit b. Let $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}_{\boldsymbol{S}}(b)$ with $b \in \{0, 1\}$, $\mathsf{CT}_0 \in \mathsf{GLWE}_{\boldsymbol{S}}(\Delta \cdot m_0)$ and $\mathsf{CT}_1 \in \mathsf{GLWE}_{\boldsymbol{S}}(\Delta \cdot m_1)$. The output ciphertext $\mathsf{CT}_{\mathsf{out}}$ is an encryption of $\Delta \cdot m_b$. The signature of the algorithm is:

$$\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CMux}(\mathsf{CT}_0, \mathsf{CT}_1, \overline{\overline{\mathsf{CT}}}).$$

Remark 3 (Public Keys) *In what follows, the public key* PUB *encompasses the bootstrapping key (*BSK*), the keyswitching key (*KSK*) and the private functional key switch key (*PFKSK*). To simplify the algorithms, we will simply refer to* PUB *when the context is clear enough to decide which key should be used.*

2.4 Representing Large Integers with TFHE

Only integer plaintexts smaller than 10 bits can be encoded in TFHE [BBB+22]. This is due to the bootstrapping, where the plaintext precision is entangled with the degree N of the cyclotomic polynomial in $\mathfrak{R}_{q,N}^k$. We briefly recall the techniques described in [BBB+22] to overcome this limitation. We first describe the encoding of large integers, and then focus on the programmable bootstrapping associated with such encoding.

Large Integer Encoding and Arithmetic Let $m \in \mathbb{Z}_p$ be the cleartext to encrypt such that $\log_2(p) > 10$. The idea is to apply a radix decomposition to m before encrypting each part into a dedicated LWE ciphertext. Let $\rho \in \mathbb{N}$ such that $2 \leq \rho \leq p < q$. Intuitively, ρ defines the message space, whereas $\frac{p}{2 \cdot \rho}$ refers to the carry space, which is getting used all along homomorphic computation. Let $\ell_z = \lceil \log_\rho(m) \rceil$. The encoded plaintext $\mathbf{pt}_{\boldsymbol{m}}$ is defined as the ρ-radix decomposition of a message $m \in \mathbb{N}$, i.e., $\mathbf{pt}_{\boldsymbol{m}} = [m_{\ell_z - 1} \cdots m_0] \leftarrow \mathsf{Encoding}^\rho(m)$ with $m = \sum_{i=0}^{\ell_z - 1} m_i \rho^i$. Then, each m_i is independently encrypted, so that $\mathbf{ct}_{\boldsymbol{m}} = [\mathsf{ct}_{m,\ell_z-1}, \cdots, \mathsf{ct}_{m,0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(m_{\ell_z - 1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(m_0)]$. Common modular integer operations are then defined on these ciphertexts,

in particular modular additions, subtractions, multiplications and division are possible.

By construction, after a given number of operations, the carry space is full and we need to call the CarryPropagate algorithm to homomorphically propagate the carries. This algorithm was first introduced in [BBB+22]. The goal of this algorithm is to clear the carry space of each input ciphertext without losing information, except for the most significant carry, which is lost. This loss occurs because, in [BBB+22], the authors work with integers modulo $\rho_\mathfrak{z}^{\ell_\mathfrak{z}}$.

When dealing with floating-point numbers, however, losing this information is not acceptable, especially when the mantissa is full. By retaining this information, we can grow the exponent if necessary. To address this limitation, we modified the CarryPropagate algorithm from [BBB+22] and named it CarryPropagateExtended, which is detailed in Algorithm 1 Note that to recover the original CarryPropagate algorithm, it is sufficient to execute the loop from 0 to $\ell_\mathfrak{z} - 2$.

In what follows, we assume $p = \rho^2$ to simplify the carry propagation. We will refer to the algorithm computing an integer operation Op as IntOp.

Programmable Bootstrapping over Large Integers (WoP-PBS) In [BBB+22], the authors present an algorithm designed for performing function evaluation on large integers and reducing the noise. The Without Padding Programmable Bootstrapping (WoP-PBS) algorithm, as described in [BBB+22], is named for its unique feature of not requiring a padding bit which distinguishes it from the typical TFHE's PBS. Having no padding bit enables the use of the total plaintext space instead of half of it, i.e., the scaling factor Δ becomes $\frac{q}{p}$ instead of $\frac{q}{2p}$ as described in Definition 1. Briefly, given a plaintext m represented using ρ bits, a WoP-PBS starts by extracting each bit of the plaintext m to obtain a list of LWEs where each one of them encrypts a bit, i.e., $\{\mathsf{ct}_i\}_{i\in[0,\dots,\rho-1]} \in \mathsf{LWE}_{\boldsymbol{s}}(m_i) \subset \mathbb{Z}_q^{(n+1)\times\rho}$. Then a circuit bootstrapping is applied to convert them into GGSW ciphertexts. The next step consists in computing a vertical packing (described in [CGGI20, Alg. 5]) to choose the right LUT depending on the value of the message. This ends by a classical blind rotation to select the correct value into the LUT. The detailed algorithm can be found in [BBB+22, Alg. 3]. The signature of the WoP-PBS is:

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{WoP\text{-}PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{LUT}_{\mathsf{f}}, \mathsf{BSK}, \mathsf{KSK}, \mathsf{PFKSK}) = \mathbf{WoP\text{-}PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{LUT}_{\mathsf{f}}, \mathsf{PUB}).$$

Algorithm 1: $\mathbf{ct}_{\mathfrak{Z}\text{out}} \leftarrow \mathsf{CarryPropagateExtended}\ (\mathbf{ct}_{\mathfrak{Z}\text{in}}, \mathsf{PUB})$

Context:
- $\Delta_{\mathfrak{Z}}$: scaling factor
- $\mathsf{LUT}_{\mathsf{Carry}}$: LUT to return the carry of the ciphertext (return $\left\lfloor \frac{m}{\beta_{\mathfrak{Z}}} \right\rfloor$).
- $\mathsf{LUT}_{\mathsf{Msg}}$: LUT to return the message of the ciphertext (return $m \mod p_{\mathfrak{Z}}$).
- $p_{\mathfrak{Z}}$: the carry-message modulus
- $\beta_{\mathfrak{Z}}$: the message modulus

Input:
- $\mathbf{ct}_{\mathfrak{Z}\text{in}} = [\mathsf{ct}_{\mathfrak{Z}\text{in},\ell_{\mathfrak{Z}}-1}, \cdots, \mathsf{ct}_{\mathfrak{Z}\text{in},0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_{\mathfrak{Z}}}$
- PUB : Public materials for **PBS-KS**

Output:
- $\mathbf{ct}_{\mathfrak{Z}\text{out}} = [\mathsf{ct}_{\mathsf{tmp}}, \mathsf{ct}_{\mathfrak{Z}\text{out},\ell_{\mathfrak{Z}}-1}, \cdots \mathsf{ct}_{\mathfrak{Z}\text{out},0}] \in \mathbb{Z}_q^{(n+1)\cdot(\ell_{\mathfrak{Z}}+1)}$

1 **for** i *in* $[0..\ell_{\mathfrak{Z}} - 1]$ **do**
  `/* Extract the carry */`
2 $\mathsf{ct}_{\mathsf{tmp}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{Z}\text{in},i}, \mathsf{PUB}, \mathsf{LUT}_{\mathsf{Carry}})$
  `/* Extract the message */`
3 $\mathsf{ct}_{\mathfrak{Z}\text{out},i} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{Z}\text{in},i}, \mathsf{PUB}, \mathsf{LUT}_{\mathsf{Msg}})$
4 **if** $i \neq \ell_{\mathfrak{Z}-1}$ **then**
5 $\mathsf{ct}_{\mathfrak{Z}\text{in},i+1} \leftarrow \mathsf{ct}_{\mathfrak{Z}\text{in},i+1} + \mathsf{ct}_{\mathsf{tmp}}$

6 **return** $(\mathbf{ct}_{\mathfrak{Z}\text{out}} = [\mathbf{ct}_{\mathsf{tmp}}, \mathsf{ct}_{\mathfrak{Z}\text{out},\ell_{\mathfrak{Z}}-1}, \cdots \mathsf{ct}_{\mathfrak{Z}\text{out},0}])$

2.4.1 Integer Subtraction

In the following floating-point algorithms, we require an operation that takes as input two vectors of ciphertexts and returns the sign of the subtraction along with a vector of ciphertexts representing the absolute value of the subtraction. This method is detailed in Algorithm 2. Intuitively, in the initial steps, an offset is added to ensure that the messages in each ciphertext of the first input are larger than those of the second input. Next, we perform the subtraction between the adjusted first input and the second input, followed by a carry propagation as described in Algorithm 1. We then extract the most significant bit (MSB) of the top block from the resulting ciphertext. Finally, we traverse all the blocks, returning the value if the MSB is set to 1, or the opposite if it is not. The sign is encoded in the padding bit, with an offset such that the most significant bit is 0 for positive values and 1 for negative values.

Lemma 1 (IntSub* (Algorithm 2)) *Let* $\mathbf{ct}_{\mathfrak{z}i} = [\mathsf{ct}_{\mathfrak{z}i,\ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}i,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{z}_{i,\ell_\mathfrak{z}-1}), \cdots \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{z}_{i,0})\right] \subseteq \mathbb{Z}_q^{(n+1)\cdot\ell_\mathfrak{z}}$ *with* $i \in \{0,1\}$ *be two ciphertexts encrypting* $\mathfrak{z}_i \in \mathbb{N}$.

Let $(\mathbf{ct}_{\mathfrak{z}\mathsf{sub}}, \mathsf{ct}_\mathfrak{s}) \leftarrow \mathsf{IntSub}^*(\mathbf{ct}_{\mathfrak{z}1}, \mathbf{ct}_{\mathfrak{z}2}, \mathsf{PUB})$. *Then* $\mathsf{Decrypt}_{\boldsymbol{s}}(\mathsf{ct}_{\mathfrak{z}\mathsf{sub}}) = |\mathfrak{z}_1 - \mathfrak{z}_2|$ *and* $\mathsf{Decrypt}_{\boldsymbol{s}}(\mathsf{ct}_\mathfrak{s}) = \mathsf{Sign}(\mathfrak{z}_1 - \mathfrak{z}_2)$.

The complexity of Algorithm 2 can be defined as $\mathbb{C}^\mathfrak{z}_{\mathsf{IntSub}^*} = (3\cdot\ell_\mathfrak{z}+1)\cdot\mathbb{C}_{\mathsf{PBS}}$

Proof 1 (Correctness of IntSub* (Algorithm 2)) *In this algorithm,* $\mathsf{CT}_\mathfrak{z}$ *can represent the mantissa or the exponent. The goal of this algorithm is to return* $|\mathfrak{z}_1 - \mathfrak{z}_2|$ *and* $\mathsf{Sign}(\mathfrak{z}_1 - \mathfrak{z}_2)$. *The first step of this algorithm is to ensure that the message* $\mathfrak{z}_1$ *is bigger than* $\mathfrak{z}_2$. *To do that, we add* $2^{(\ell_\mathfrak{z}+1)\cdot\rho_\mathfrak{z}-1}$ *to* $\mathfrak{z}_1$, *which automatically guarantees that all the* LWE *that compose* $\mathfrak{z}_1$ *such that all the* LWE *of* $\mathfrak{z}_1$ *are bigger than the ones of* $\mathfrak{z}_2$. *Now we can perform the subtraction term by term. After a carry propagate, if* $\mathfrak{z}_1 > \mathfrak{z}_2$, *we retrieve on the most significant* LWE *the added value at the beginning of the algorithm. Otherwise, this added value is used during the subtraction. This value corresponds to* $\mathsf{Sign}(\mathfrak{z}_1 - \mathfrak{z}_2)$. *By extracting this sign, and adding it to each* LWE, *with a* PBS, *we can see whether* $\mathfrak{z}_1 > \mathfrak{z}_2$ *or not and choose between the ciphertext obtain after the subtraction or its opposite and get* $|\mathfrak{z}_1 - \mathfrak{z}_2|$.

Lemma 2 (Noise Constraints of Algorithm 2) *The output noise variances of ciphertexts of Algorithm 2,* $\mathsf{ct}_\mathfrak{s}$ *and* $\mathsf{ct}_{\mathfrak{z}\mathsf{sub}}$, *are respectively* $4\cdot\sigma^2_{\mathsf{BR}}$ *and* σ^2_{BR}.

To guarantee correctness of this operation, we need to find parameters that verify the following inequalities:

$$2\cdot\sigma^2_{\mathsf{in}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2.$$

with σ_{BR} *the noise added by the blind rotation,* σ_{KS} *the noise added by the keyswitch,* σ_{cmux} *the noise added by the* CMux *and finally* σ_{MS}, *the noise added by the modulus switch.*

Proof 2 (Proof of Lemma 2) *Let us look at the noise propagation in Algorithm 2. We assume that each input ciphertext contains a noise following a centered Gaussian distribution with a variance* σ^2_{in}. *These noises are also assumed to be statistically independent.*

At the end of line 6, the variance of the noise in $\mathsf{ct}_{\mathfrak{z}\mathsf{sub},i}$ *is* $2\cdot\sigma^2_{\mathsf{in}}$ *for* $0 \leq i < l_\mathfrak{z}$. *The worst operation in terms of noise in the carry propagation*

Algorithm 2: $(\mathbf{ct}_{\mathfrak{z}_{sub}}, \mathsf{ct}_{\mathfrak{s}}) \leftarrow \mathsf{IntSub}^{*}\ (\mathbf{ct}_{\mathfrak{z}1}, \mathbf{ct}_{\mathfrak{z}2}, \mathsf{PUB})$

Context:
- $\Delta_{\mathfrak{z}}$: scaling factor
- $\mathsf{LUT}_{\mathsf{Extract}}$: LUT to extract the MSB (i.e. the $(\log_2(q)-2)^{th}$bit)
- $\mathsf{LUT}_{\mathsf{f}}$: LUT to return $\mathfrak{z} - q/4$ if the MSB equals 1; $q/4 - \mathfrak{z}$ otherwise
- $p_{\mathfrak{z}}$: the carry-message modulus
- $\beta_{\mathfrak{z}}$: the message modulus

Input:
- $\mathbf{ct}_{\mathfrak{z}1} = [\mathsf{ct}_{\mathfrak{z}1,\ell_{\mathfrak{z}}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}1,0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_{\mathfrak{z}}}$
- $\mathbf{ct}_{\mathfrak{z}2} = [\mathsf{ct}_{\mathfrak{z}2,\ell_{\mathfrak{z}}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}2,0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_{\mathfrak{z}}}$
- PUB : Public materials for **PBS-KS** and for CBS

Output:
- $\mathbf{ct}_{\mathfrak{z}_{\mathsf{sub}}} = [\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1}, \cdots \mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_{\mathfrak{z}}}$
- $\mathsf{ct}_{\mathfrak{s}} \in \mathbb{Z}_q^{n+1}$

1 **for** i *in* $[1..\ell_{\mathfrak{z}}-1]$ **do**
  `/* Ensure that` $\mathsf{ct}_{\mathfrak{z}1,i}$ `is larger than` $\mathsf{ct}_{\mathfrak{z}2,i}$ `*/`
2 $\mathsf{ct}_{\mathfrak{z}1,i} \leftarrow \mathsf{ct}_{\mathfrak{z}1,i} + \mathsf{TrivialEncrypt}(2^{2\cdot\rho_{\mathfrak{z}}-1}, 1)$
3 $\mathsf{ct}_{\mathfrak{z}1,i} \leftarrow \mathsf{ct}_{\mathfrak{z}1,i} - \mathsf{TrivialEncrypt}(2^{\rho_{\mathfrak{z}}-1}, 1)$
4 $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i} \leftarrow \mathsf{ct}_{\mathfrak{z}1,i} - \mathsf{ct}_{\mathfrak{z}2,i}$
5 $\mathsf{ct}_{\mathfrak{z}1,0} \leftarrow \mathsf{ct}_{\mathfrak{z}1,0} + \mathsf{TrivialEncrypt}(2^{2\cdot\rho_{\mathfrak{z}}-1}, 1)$
6 $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},0} \leftarrow \mathsf{ct}_{\mathfrak{z}1,0} - \mathsf{ct}_{\mathfrak{z}2,0}$
7 $\mathbf{ct}_{\mathfrak{z}_{sub}} \leftarrow \mathsf{CarryPropagate}(\mathbf{ct}_{\mathfrak{z}_{sub}} = [\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},0}], \mathsf{PUB})$
  `/* Extract the msb to get the sign */`
8 $\mathsf{ct}_{\mathfrak{s}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1}, \mathsf{PUB}, \mathsf{LUT}_{\mathsf{Extract}})$
  `/* Return the value if MSB==1, the opposite otherwise */`
9 **for** i *in* $[0..\ell_{\mathfrak{z}}-1)$ **do**
10 $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i} \leftarrow \mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i} + \mathsf{ct}_{\mathfrak{s}}$
11 $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i}, \mathsf{PUB}, \mathsf{LUT}_{\mathsf{f}})$
12 $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1}, \mathsf{PUB}, \mathsf{LUT}_{\mathsf{f}})$
  `/* Put the sign on the padding bit plus flip the bit to keep the representation 0 is positive and 1 is negative */`
13 $\mathsf{ct}_{\mathfrak{s}} \leftarrow \mathsf{ct}_{\mathfrak{s}} \cdot 2 + \mathsf{TrivialEncrypt}(q/2, 1)$
14 **return** $(\mathsf{ct}_{\mathfrak{s}}, \mathbf{ct}_{\mathfrak{z}_{sub}} = [\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},\ell_{\mathfrak{z}}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},0}])$

on line 7 consists of adding a freshly bootstrapped ciphertext with one of the $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}},i}$ *and applying to it a keyswitch and a bootstrapping. It means that to*

have correctness we must verify the following inequality:

$$2 \cdot \sigma_{\mathsf{in}}^2 + \sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2.$$

with $\sigma_{\mathsf{BR}}^2, \sigma_{\mathsf{KS}}^2, \sigma_{\mathsf{MS}}^2$*, the noise after respectively a bootstrapping, a keyswitch and a modulus switch and* t^2*, the noise bound as defined in the proof of the noise constraints of Algorithm 6.*

On line 8, another noise constraint appears.

$$\sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2.$$

As the left hand term is smaller than the one of the previous inequality, we can discard this constraint. On lines 11 and 12, we have the following constraints

$$2 \cdot \sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2 \ \& \ \sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2.$$

Using the fact that the last constraint is dominated by the others, we can remove it from the set of constraints. In the end of Algorithm 2, the sign $\mathsf{ct}_{\mathfrak{s}}$ *has a noise variance of* $4 \cdot \sigma_{\mathsf{BR}}^2$ *and the each ciphertext in the vector* $\mathsf{ct}_{\mathfrak{z}_{\mathsf{sub}}}$ *has a noise variance* σ_{BR}^2*.*

2.5 Traditional Floating-Point Representation

Floating-points have become the standard to represent real numbers in computer science, as described in [Ins08]. Their main advantage lies in having a variable precision all along computations, giving more flexibility and accuracy. Usually a floating-point is represented by three values: the sign, the mantissa and the exponent. The most common floating-point encodings on CPU are the single precision, represented on 32 bits (with 1 bit of sign, 8 bits of exponent and 23 bits of mantissa) and the double precision over 64 bits (1 bit of sign, 11 bits of exponent and 52 bits of mantissa). Less common but still useful encodings are the half-precision which represents 16-bit floats, or some alternative called `Bfloat` [WK19]. Without getting into details, these encodings mainly differ in the distribution of the bit number associated to the mantissa and the exponent. Finally, another family called `MiniFloat` is dedicated to floats whose the size is 8 bits. We refer to [MBDD+18] for more information about floats.

Definition 4 (Floating-Point) *Let* $\mathfrak{b} \in \mathbb{N}$ *such that* $\mathfrak{b} \geq 2$. *Let* $max_{\mathfrak{e}} \in \mathbb{N}^*$ *and a fixed* $\mathsf{bias} \in [0, max_{\mathfrak{e}}]$. *A floating-point number* $x \in \mathbb{R}$ *is partially characterized by three values* $(\mathfrak{s}, \mathfrak{m}, \mathfrak{e}) \in \{0;1\} \times \mathbb{N} \times [0, max_{\mathfrak{e}}]$, *such that:* $x = (-1)^{\mathfrak{s}} \cdot \mathfrak{m} \cdot \mathfrak{b}^{\mathfrak{e}-\mathsf{bias}}$. *With this definition, a floating-point number may have several representations. To obtain a unique representation, the mantissa* $\mathfrak{m}$ *must be in the interval* $[1, \mathfrak{b})$ *except for the value zero where* $\mathfrak{m} = 0$.

3 Homomorphic Floating-Points (HFP)

In this section, we present a promising but limited approach in terms of precision which leverages the WoP-PBS to efficiently compute any operation over floating-point numbers. Then, we explain how to efficiently translate the traditional representation of floating points into a TFHE-friendly way for larger precisions. Finally, we describe the first building blocks needed to perform higher-level operations on these homomorphic floating-points.

3.1 MiniFloats: WoP-PBS Based Floats

A powerful approach to defining homomorphic floats for TFHE-like schemes relies on the WoP-PBS. The method is somewhat similar to the gate bootstrapping approach defined in [CGGI20]: almost every operation is performed using a WoP-PBS.

Minifloat Encoding Let ρ be the number of bits of precision for a message in an LWE ciphertext, and let $\rho_{\mathfrak{m}}$ (resp. $\rho_{\mathfrak{e}}$) be the number of bits of the mantissa (resp. the exponent). In this first attempt at building TFHE-minifloats, we do not need to have distinct ciphertexts for the mantissa, the sign and the exponent. For instance, we can define an 8-bit floating-point with $\rho = 4$, $\rho_{\mathfrak{m}} = 3$ and $\rho_{\mathfrak{e}} = 4$ using only two LWEs, $\mathsf{ct}_1 \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}||\mathfrak{m}_2||\mathfrak{m}_1||\mathfrak{m}_0)$ and $\mathsf{ct}_2 \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_3||\mathfrak{e}_2||\mathfrak{e}_1||\mathfrak{e}_0)$ where $\mathfrak{m}_i$ (resp. $\mathfrak{e}_i$) corresponds to the i^{th} bit of the mantissa (resp. of the exponent). Each element can be dispatched in any order, but the order must be publicly known to correctly generate the LUT. We call this encoding the *minifloat encoding* and we write it as follows:

$$\text{TFHE-Minifloat}^{\rho}(\rho_{\mathfrak{m}}, \rho_{\mathfrak{e}}, \mathsf{bias}) = \mathsf{Encoding}^{\rho}(\mathfrak{s}||\mathfrak{m}||\mathfrak{e}||).$$

Minifloat Operations Defining operations over the minifloat encoding is easy: each one of them is computed with a WoP-PBS where the LUT associated with the operation is given. The WoP-PBS can easily be extended to take many ciphertexts as input in order to compute multivariate operations: the bit extraction step can be done for every input and then a single CMux tree using the bits of every input. Let Op be an operation (e.g., an addition), and $\mathsf{LUT}_{\mathsf{Op}}$ its associated LUT (for more details on how to properly generate the LUT, we refer to [BBB+22]). For some $k \in \mathbb{N}$, let $\mathbf{ct}_{\mathfrak{f}_i}$ for $i \in [0, k-1]$ be the input ciphertexts. Then, the output is given by: $\mathbf{ct}_{\mathfrak{f}_{\mathsf{out}}} \leftarrow \mathbf{WoP\text{-}PBS}(\{\mathbf{ct}_{\mathfrak{f}_i}\}_{i=0}^{k-1}, \mathsf{LUT}_{\mathsf{Op}}, \mathsf{PUB})$.

The main advantage is about the complexity of this method, which is not dependent on the functions that have to be computed, i.e., any univariate functions will take the same time (e.g., cosine, logarithm, ...).

Remark 4 *Some operations do not require a complete WoP-PBS. For example, to perform a ReLU, we only need to extract the sign and perform a* CMux *between input value and a trivial* LWE *(defined in 4) that encrypts zero.*

We provide benchmarks for this method in Section 6. This method is very efficient but it is limited in terms of precision. In fact, this method does not work when the combined bit size of all inputs of a WoP-PBS exceeds approximately twenty bits, because the number of values that need to be represented is too large and the LUT quickly becomes too big [BBB+22, Remark 8]. Next section explores another encoding that can efficiently support large floating-point numbers.

3.2 Homomorphic Floating-Point Encoding

As in the traditional representation of floating-point numbers, the homomorphic floating point representation is divided into three parts: the sign, the mantissa and the exponent.

The **sign** ($\mathfrak{s}$) is encoded by one LWE ciphertext. This ciphertext encrypts the value 0 if the sign is positive or 1 if the sign is negative.

The **mantissa** ($\mathfrak{m}$) is encoded by several LWE ciphertexts (at least 2). Each ciphertext associated with the mantissa encodes the same amount of message bits (denoted $\rho_{\mathfrak{m}}$ in the following). For a mantissa represented by $\ell_{\mathfrak{m}}$ LWE ciphertexts, we can represent integers in $\left[0; 2^{\ell_{\mathfrak{m}} \cdot \rho_{\mathfrak{m}}}\right)$. The ciphertext encrypting the most significant bits (respectively, the least significant bits) of

the mantissa is called the most significant (respectively, the least significant) ciphertext. With this representation, we can ensure that the precision of the mantissa is at least $((\ell_{\mathfrak{m}} - 1) \cdot \rho_{\mathfrak{m}} + 1)$-bits. Indeed, the least significant ciphertext will be discarded after some operations, so the information in this block must not be seen as additional precision: when the carry space of the most significant LWE ciphertext is full, a new LWE ciphertext is added as the new most significant block. The less significant block is then removed and the exponent value is increased. In floating point arithmetic, this approach is generally called *rounding towards zero*. This means that the least significant bits from the mantissa are discarded. This way, the exponent can be smaller and represent a large range of values. In our representation, to keep a unique encoding for any floating-point, the most significant block should always be different from zero (except for the special value zero where all the blocks are equal to zero). So for any non-zero values, the mantissa is an integer in $\left[2^{\rho_{\mathfrak{m}} \cdot (\ell_{\mathfrak{m}} - 1)}; 2^{\rho_{\mathfrak{m}} \cdot \ell_{\mathfrak{m}}}\right)$.

The **exponent** ($\mathfrak{e}$) is encoded by one or more LWE ciphertexts. Each LWE ciphertext encrypts the same amount of bits $\rho_{\mathfrak{e}}$. The value represented by the exponent is in base $2^{\rho_{\mathfrak{m}}}$ (as already mentioned in the mantissa). So an exponent encrypted in $\ell_{\mathfrak{e}}$ LWE ciphertexts represents values in $\left[0; (2^{\rho_{\mathfrak{m}}})^{2^{\ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}}}}\right)$. To represent an exponent that can be negative or positive, the positive value encoded in these $\ell_{\mathfrak{e}}$ LWE ciphertexts needs to be subtracted by a value named bias. When we decode, we obtain, $e \in \left[(2^{\rho_{\mathfrak{m}}})^{-\mathsf{bias}}; (2^{\rho_{\mathfrak{m}}})^{2^{\ell \cdot \rho_{\mathfrak{e}}} - \mathsf{bias}}\right)$. The **encoding of the TFHE floating-point** is illustrated in Figure 1, and we refer to it as follows:

$$\mathsf{TFHE\_FP}(\ell_{\mathfrak{m}}, \rho_{\mathfrak{m}}, \ell_{\mathfrak{e}}, \rho_{\mathfrak{e}}, \mathsf{bias}).$$

Bias The value bias can be set to any value, but to represent a large range of values, in the traditional floating-point, this value is often set to be half of the range of the exponent. With our representation, this value should correspond to $2^{\ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}} - 1}$, but in our algorithm we choose to use $\mathsf{bias} = 2^{\ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}} - 1} + \ell_{\mathfrak{m}} - 1$. Through this specific value, we gain a speed-up in the homomorphic floating multiplication proposed in Algorithm 10.

Special Values In the floating-point arithmetic, the subnormal values are the closest values to zero: they are represented by an exponent equal to zero and the leading significant digits equal to zero. In our implementation

we choose to not represent this values to have better performance, but our algorithm can be easily modified to take these values into account. In what follows, the leading significant block is always strictly positive except for the zero value. This is the only value represented by each mantissa and exponent blocks equals to zero. Thus, if an operation yields an encrypted float which has its most significant LWE equal to zero, the result will encrypt the zero value (i.e., all the mantissa and exponent LWE will be equal to zero). To keep the algorithms easier to read, other special values like infinities, or NaN are voluntary excluded. Note that the process to support these is nevertheless detailed in Section 7.1.

Encoding and Encrypting We propose an algorithm to encode and decode real numbers for TFHE with the representation $\mathsf{TFHE\_Fp}(\ell_{\mathfrak{m}}, \rho_{\mathfrak{m}}, \ell_{\mathfrak{e}}, \rho_{\mathfrak{e}}, \mathsf{bias})$. Let $\mathfrak{f}$ be a real number. First, we need to find $\mathfrak{m} \in [0, \ell_{\mathfrak{m}} \cdot \rho_{\mathfrak{m}})$ and $\mathfrak{e} \in [0, \ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}})$ such that:

$$\mathfrak{f} = (-1)^{\mathfrak{s}} \cdot \mathfrak{m} \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}-\mathsf{bias}}.$$

To obtain a unique representation of these floating-point representations, we impose that the most significant block of the mantissa must be strictly positive (except for the value zero). From this first encoding, we split the mantissa and the exponent according to the $2^{\rho_{\mathfrak{m}}}$ and $2^{\rho_{\mathfrak{e}}}$-radix decompositions, i.e., $\mathfrak{m} \leftarrow \mathsf{Encoding}^{2^{\rho_{\mathfrak{m}}}}(\mathfrak{m})$ and $\mathfrak{e} \leftarrow \mathsf{Encoding}^{2^{\rho_{\mathfrak{e}}}}(\mathfrak{e})$. The final encoding is given by:

$$\mathfrak{f} = (\mathfrak{s}, \mathfrak{m} = (\mathfrak{m}_{\ell_{\mathfrak{m}}-1}, \cdots, \mathfrak{m}_0)_{\rho_{\mathfrak{m}}}, \mathfrak{e} = (\mathfrak{e}_{\ell_{\mathfrak{e}}-1}, \cdots, \mathfrak{e}_0)_{\rho_{\mathfrak{e}}}).$$

In absolute value, the maximum value represented by this encoding is $\max = (2^{\ell_{\mathfrak{m}} \cdot \rho_{\mathfrak{m}}} - 1) \cdot (2^{\rho_{\mathfrak{m}}})^{2^{\ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}}} - \mathsf{bias} - 1}$. Since the subnormal values are not taken into account, the minimum positive value reached by this encoding (without zero) is $\min = (2^{\ell_{\mathfrak{m}} \cdot \rho_{\mathfrak{m}} - 1}) \cdot (2^{\rho_{\mathfrak{m}}})^{-\mathsf{bias}}$. In Algorithm 3 (resp. Algorithm 4), the method to encrypt (resp. decrypt) a floating-point number is described. The following lemma states the correctness and the notations used to represent homomorphic floats.

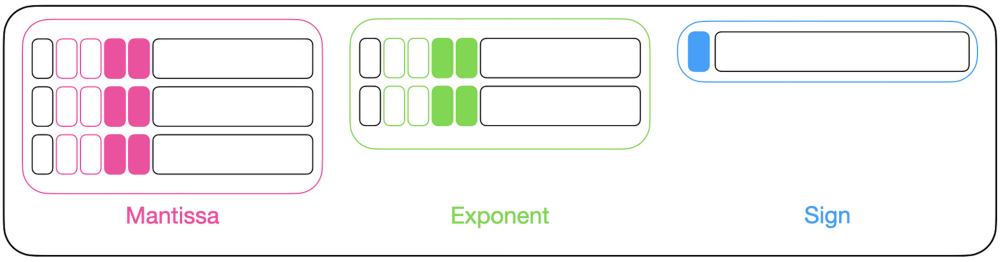

Figure 1: The figure illustrates the encoding of a homomorphic floating-point. The Mantissa (in pink) and the Exponent (in green) are split in several ciphertexts which each ciphertext encrypting 4 bits. The fully colored boxes in the figure represent the bits of messages space and the empty colored boxes correspond to the bits of carry space. The sign (in blue) is encoded in only one ciphertext where the information is encrypted on the most significant bit.

Algorithm 3: $\mathbf{ct}_\mathfrak{f} \leftarrow \mathsf{EncryptFloat}(\mathfrak{s}, \mathbf{m}, \mathfrak{e})$

Input: $\mathsf{EncodeFloat}(x) = (\mathfrak{s}, \mathbf{m}, \mathfrak{e})$
Output: $\mathbf{ct}_\mathfrak{f} = [\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_\mathfrak{m}, \mathbf{ct}_\mathfrak{e}]$
1 $\mathsf{ct}_\mathfrak{s} \leftarrow \mathsf{Encrypt}_{\boldsymbol{s}}(\frac{q}{2} \cdot \mathfrak{s}) \in \mathsf{LWE}(\mathfrak{s})$
2 $\mathbf{ct}_\mathfrak{m} = (\mathsf{ct}_{\mathfrak{m},\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}) \leftarrow \mathsf{Encrypt}_{\boldsymbol{s}}(\mathbf{m})$
3 $\mathbf{ct}_\mathfrak{e} = (\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}) \leftarrow \mathsf{Encrypt}_{\boldsymbol{s}}(\mathfrak{e})$
4 **return** $\mathbf{ct}_\mathfrak{f} = [\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_\mathfrak{m}, \mathbf{ct}_\mathfrak{e}]$

Algorithm 4: $\mathfrak{f} \leftarrow \mathsf{DecryptFloat}(\mathsf{ct}_\mathfrak{f})$

Input: $\mathbf{ct}_\mathfrak{f} = [\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_\mathfrak{m}, \mathbf{ct}_\mathfrak{e}]$
Output: $\mathfrak{f} = (-1)^\mathfrak{s} \cdot \mathfrak{m} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}-\mathsf{bias}} \in \mathbb{R}$
1 **If** $\mathsf{Decrypt}_{\boldsymbol{s}}(\mathsf{ct}_\mathfrak{s}) = 0$ **then** $\mathfrak{s} = 1$, **Else** $\mathfrak{s} = -1$
2 $\mathfrak{m} \leftarrow \mathsf{Decrypt}_{\boldsymbol{s}}(\mathsf{ct}_\mathfrak{m})$
3 $\mathfrak{e} \leftarrow \mathsf{Decrypt}_{\boldsymbol{s}}(\mathsf{ct}_\mathfrak{e})$
4 **return** $\mathfrak{f} \leftarrow (-1)^\mathfrak{s} \cdot \mathfrak{m} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}-\mathsf{bias}}$

Lemma 3 (Correctness of DecryptFloat (4)) *Let* $\mathfrak{f} = (\mathfrak{s}, \mathfrak{m}, \mathfrak{e}) \in \{0,1\} \times [0, \ell_\mathfrak{m} \cdot \rho_\mathfrak{m}) \times [0, \ell_\mathfrak{e} \cdot \rho_\mathfrak{e})$. *Let* $\mathsf{ct}_\mathfrak{f} \leftarrow \mathsf{EncryptFloat}(\mathfrak{f})$ *such that* $\mathsf{ct}_\mathfrak{f} = [\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_\mathfrak{e}, \mathbf{ct}_\mathfrak{m}]$, *with* $\mathsf{ct}_\mathfrak{s} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s})$, $\mathbf{ct}_\mathfrak{e} = [\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1_{\ell_\mathfrak{e}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1_0}\right)\right]$ *and* $\mathbf{ct}_\mathfrak{m} = [\mathsf{ct}_{\mathfrak{m},\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1_{\ell_\mathfrak{m}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1_0}\right)\right]$. *Then,* $\mathsf{DecryptFloat}_{\boldsymbol{s}}(\mathsf{ct}_\mathfrak{f}) = \mathfrak{f}$.

Trivial Encrypt A trivial encryption is an LWE ciphertext where all the a_i are equal zero (for $i \in [0, n)$). This is trivially extendable to the floats. This is denoted $\mathsf{TrivialEncryptFloat}(\mathfrak{f})$. Sometimes, we only need to trivially encrypt a part of a floating-point: which is suggested by the notation $\mathsf{TrivialEncrypt}(\mathsf{Value}, \mathsf{NumberOfBlocks})$. For instance, to encrypt the value $\mathfrak{v}$ as an exponent, we note $\mathsf{TrivialEncrypt}(\mathfrak{v}, \ell_\mathfrak{e})$.

Definition 5 (Maximum error after operation) *In the context of floating-point, due to the encoding, after each operation a small error could be introduced (this error is not tied to the TFHE noise). This added error is denoted ϵ. To find the errors added after each operation by our encoding, we look at the maximal error added in the mantissa and we multiply this error by the exponent. For a floating-point $\mathfrak{f} = (-1)^{\mathfrak{s}} \cdot \mathfrak{m} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}-\mathsf{bias}}$, the error after an operation can be bounded by $\mathsf{error}_\mathfrak{m} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}-\mathsf{bias}}$. As we do not represent the subnormal values, if the $\mathfrak{e}$ is equal to 0, the error is bounded by $2^{\rho_\mathfrak{m} \cdot (\ell_\mathfrak{m}-1)} \cdot (2^{\rho_\mathfrak{m}})^{-\mathsf{bias}}$.*

Example: Encoding a 64 bits Floating-Point In the [Ins08] standard, a floating-point is composed of 1 bit of sign, 11 bits of exponent and 52 bits plus one hidden bit of mantissa. So as mentioned in the beginning of Section 3, to ensure a precision of at least 53 bits, we need to have $(\ell_\mathfrak{m} - 1) \cdot \rho_\mathfrak{m} + 1 \geq 53$ i.e., one additional block to perform operations without losing the precision of 53 bits. For the mantissa, we choose $\ell_\mathfrak{m} = 27$ with $\rho_\mathfrak{m} = 2$. In [Ins08], the exponent value $\mathfrak{e}$ belongs to $[-1023, 1024)$. To simplify the implementation, we prefer to have $\rho_\mathfrak{e} = \rho_\mathfrak{m}$, and a bias equal to $2^{\ell_\mathfrak{e} \cdot \rho_\mathfrak{e} - 1} + \ell_\mathfrak{m} - 1$ with $\ell_\mathfrak{e} = 5$. Thus, this yields in a floating-point exponent in $\left[-\mathsf{bias} \cdot \rho_\mathfrak{m}, (2^{\ell_\mathfrak{e} \cdot \rho_\mathfrak{e}} - \mathsf{bias} - 1) \cdot \rho_\mathfrak{m}\right] = [-1076, 970]$ with $\mathsf{bias} = 538$. This allows representing as many values as the standard one, but with a different scale: the upper bound is lower, but more precision is given near values close to zero. These parameters correspond to the representation :

$$\mathsf{TFHE\_Fp_{64b}}(\ell_\mathfrak{m} = 27, \rho_\mathfrak{m} = 2, \ell_\mathfrak{e} = 5, \rho_\mathfrak{e} = 2, \mathsf{bias} = 538).$$

More encoding examples for 32-bits, 16-bits and 8-bits are given in Table 3 (Section 6).

3.3 Choosing Between Two Ciphertexts

In what follows, we extensively use Algorithm 5 to homomorphically make a choice between two LWE ciphertext lists depending on an encrypted bit. This algorithm is an extended version of the CMux described in [CGGI20, Lemma 3.16]. The selector is a GGSW ciphertext, and the choice is done between two lists of LWE ciphertexts.

Lemma 4 *Let $\mathbf{ct}_i = [\mathsf{ct}_{\mathfrak{z}_i, \ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{\mathfrak{z}_i,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{z}_{i,\ell_\mathfrak{z}-1}), \cdots \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{z}_{i,0})\right] \subseteq \mathbb{Z}_q^{(n+1)\cdot\ell_\mathfrak{z}}$ with $i \in \{0,1\}$ be two ciphertexts encrypting $\mathfrak{z}_i \in \mathbb{N}$. Let $\overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}} \in$*

Algorithm 5: $\mathbf{ct}_{\mathsf{res}} \leftarrow \mathsf{ExtendedCMux}(\mathbf{ct}_0, \mathbf{ct}_1, \overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}})$

Input:
- $\mathbf{ct}_0 = [\mathsf{ct}_{0,\ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{0,0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_\mathfrak{z}}$ with $\mathsf{ct}_{i,j} \in \mathsf{LWE}_{\boldsymbol{s}}(m_{i,j})$ for some plaintext $m_{i,j}$
- $\mathbf{ct}_1 = [\mathsf{ct}_{1,\ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{1,0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_\mathfrak{z}}$
- $\overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}} \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(b)$ (with $b \in \{0,1\}$)

Output:
- $\mathbf{ct}_{\mathsf{res}} = [\mathsf{ct}_{b,\ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{b,0}] \in \mathbb{Z}_q^{(n+1)\cdot\ell_\mathfrak{z}}$

1 **for** i *in* $[0..\ell_\mathfrak{z})$ **do**
2 $\quad \mathsf{CT}_{0,i} \leftarrow \mathsf{SampleInsert}(\mathsf{ct}_{0,i})$, $\mathsf{CT}_{1,i} \leftarrow \mathsf{SampleInsert}(\mathsf{ct}_{1,i})$
3 $\quad \mathsf{CT}_{\mathsf{res},i} \leftarrow \mathsf{CMux}(\mathsf{CT}_{0,i}, \mathsf{CT}_{1,i}, \overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}})$
4 $\quad \mathsf{ct}_{\mathsf{res},i} \leftarrow \mathsf{Sample\_extract}(\mathsf{CT}_{\mathsf{res},i})$
5 **return** $\mathbf{ct}_{\mathsf{res}} = [\mathsf{ct}_{\mathsf{res},\ell_\mathfrak{z}-1}, \cdots, \mathsf{ct}_{\mathsf{res},0}]$

$\mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(b)$ *(with* $b \in \{0,1\}$*).* *Let* $\mathbf{ct}_{\mathsf{res}} \leftarrow \mathsf{ExtendedCMux}(\mathbf{ct}_0, \mathbf{ct}_1, \overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}})$. *Then,* $\mathsf{Decrypt}_{\boldsymbol{s}}(\mathbf{ct}_{\mathsf{res}}) = \mathfrak{z}_\mathfrak{b}$.

Proof 3 (Correctness of Algorithm 5) *Let us execute each instruction of Algorithm 5. In Lines 2 and 3, both* LWE *ciphertexts are transformed into* GLWE *ciphertexts with the message on the first coefficient and random messages on all the other coefficients, i.e.,* $\mathsf{CT}(M_{j,i}) \in \mathsf{GLWE}_{\boldsymbol{S}}(M_{j,i})$ *with* $M_{j,i} = m_{j,i} + \sum_{\alpha=1}^{N-1} r_{j,i,\alpha} X^\alpha$*, for some* $r_{i,j,\alpha} \in \mathbb{Z}_q$ *with* $j \in \{0,1\}$*. Next, in Line 4:* $\mathsf{CT}_{res,i}$ *is the result of the* CMux*, i.e.,* $\mathsf{CT}_{res,i}(M_{b,i}) \leftarrow \mathsf{CMux}\left(\mathsf{CT}(M_{1,i}), \mathsf{CT}(M_{0,i}), \overline{\overline{\mathsf{CT}}}_{\mathsf{Sel}}(b)\right)$*, (as define in 2.3), with* $b \in \{0,1\}$*. At Line 5, we extract the first coefficient of* $\mathsf{CT}_{m_{res},i}$*. The result is then* $\mathsf{ct}(m_{res,i}) \in \mathsf{LWE}_{\boldsymbol{s}}(m_{b,i})$.

Proof 4 (CMux Noise Analysis) *We adapt [CLOT21, Proof 11] for an external product using a binary secret in the GGSW ciphertext. The only difference is that the GGSW ciphertext is obtained through a* CBS *rather than a freshly encrypted ciphertext, so we use the same noise formula. However, the noise of the bootstrapping key is defined as the output noise of circuit bootstrapping instead of fresh encryption.*

3.4 Propagating the Carries

Since HFP are based on radix-based homomorphic integers, the need to propagate the carry must be considered to ensure correctness. Indeed, in each

block, carry might accumulate all along computation up to point where the carry space is full. Differently from modular integer computation, where the modulus is generally a power of two, we cannot simply remove the carry from the most significant block. In our case, when the mantissa has a carry that has been propagated to the most significant block, a new one must be created and the last one can be removed. In Algorithm 6, we describe the process to perform this homomorphically. It takes as input a ciphertext encrypting a floating-point, and returns another ciphertext where the carries have been propagated.

Algorithm 6: $\mathbf{ct}_{\mathfrak{f}_{\text{out}}} \leftarrow \mathsf{CarryPropagateFloat}\left(\mathbf{ct}_{\mathfrak{f}}\right)$

Context: $\mathsf{LUT}_{\mathsf{id}}$: Lookup Table associated to the id function.

Input:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}} = \begin{bmatrix} \mathsf{ct}_{\mathfrak{s}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}\right) \\ \mathbf{ct}_{\mathfrak{e}} = \left[\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)\right] \\ \mathbf{ct}_{\mathfrak{m}} = \left[\mathsf{ct}_{\mathfrak{m},\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\ell_{\mathfrak{m}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_0\right)\right] \end{bmatrix} \\ \mathsf{PUB} : \text{Public keys for KS-PBS and for CBS} \end{cases}$$

Output:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}_{\text{out}}} = \begin{bmatrix} \mathsf{ct}_{\mathfrak{s}_{\text{out}}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}\right) \\ \mathbf{ct}_{\mathfrak{e}_{\text{out}}} = \left[\mathsf{ct}_{\mathfrak{e}_{\text{out}},\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_{\text{out}},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\text{out},\ell_{\mathfrak{e}}-1,}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\text{out},0}\right)\right] \\ \mathbf{ct}_{\mathfrak{m}_{\text{out}}} = \left[\mathsf{ct}_{\mathfrak{m}_{\text{out}},\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_{\text{out}},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\text{out},\ell_{\mathfrak{m}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\text{out},0}\right)\right] \end{bmatrix} \end{cases}$$

1 $\mathsf{ct}_{\mathfrak{s}_{\text{out}}} \leftarrow \mathsf{KS\text{-}PBS}\left(\mathsf{ct}_{\mathfrak{s}}, \mathsf{LUT}_{\mathsf{Id}}, \mathsf{PUB}\right)$

2 $\mathbf{ct}_{\mathfrak{e}'} \leftarrow \mathsf{CarryPropagate}\left(\mathbf{ct}_{\mathfrak{e}}, \mathsf{PUB}\right)$

3 $\mathbf{ct}_{\mathfrak{m}'} = \left[\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}, \cdots, \mathsf{ct}_{\mathfrak{m}',0}\right] \leftarrow \mathsf{CarryPropagateExtend}\left(\mathbf{ct}_{\mathfrak{m}}, \mathsf{PUB}\right)$; `/* Algo 1 */`

4 $\mathbf{ct}_{\mathfrak{m}_+} = \left[\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}, \cdots, \mathsf{ct}_{\mathfrak{m}',1}\right]$, $\mathbf{ct}_{\mathfrak{m}_-} = \left[\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}',0}\right]$

5 $\overline{\overline{\mathsf{CT}}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{\mathfrak{m}'_{\ell_{\mathfrak{m}}}}, \mathsf{PUB}\right)$

6 $\mathbf{ct}_{\mathfrak{m}_{\text{out}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_-}, \mathbf{ct}_{\mathfrak{m}_+}, \overline{\overline{\mathsf{CT}}}\right)$,
$\mathbf{ct}_{\mathfrak{e}_{\text{out}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}'}, \mathbf{ct}_{\mathfrak{e}'} + \mathsf{TrivialEncrypt}(1, \ell_{\mathfrak{e}}), \overline{\overline{\mathsf{CT}}}\right)$; `/* Algo 5 */`

7 **return** $\mathbf{ct}_{\mathfrak{f}_{\text{out}}} = \left(\mathsf{ct}_{\mathfrak{s}_{\text{out}}}; \mathbf{ct}_{\mathfrak{e}_{\text{out}}}; \mathbf{ct}_{\mathfrak{m}_{\text{out}}}\right)$

Lemma 5 *Let* $\mathbf{ct}_{\mathfrak{f}} = \left[\mathsf{ct}_{\mathfrak{s}}, \mathbf{ct}_{\mathfrak{m}}, \mathbf{ct}_{\mathfrak{e}}\right]$ *with* $\mathsf{ct}_{\mathfrak{s}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}\right)$, $\mathbf{ct}_{\mathfrak{e}} = \left[\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)\right]$, $\mathbf{ct}_{\mathfrak{m}} = \left[\mathsf{ct}_{\mathfrak{m},\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\ell_{\mathfrak{m}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_0\right)\right]$ *such that some* $\mathsf{ct}_{\mathfrak{m},i}$ *(resp.* $\mathsf{ct}_{\mathfrak{e},i}$*) encrypting some* $\mathfrak{m}_i > 2^{\rho_{\mathfrak{m}}}$ *(resp.* $\mathfrak{e}_i > 2^{\rho_{\mathfrak{e}}}$*). Let* $\mathbf{ct}_{\mathfrak{f}_{\text{out}}} \leftarrow \mathsf{CarryPropagateFloat}\left(\mathbf{ct}_{\mathfrak{f}}\right)$. *Let* $\mathfrak{f} = \mathsf{DecryptFloat}_{\boldsymbol{s}}(\mathbf{ct}_{\mathfrak{f}}) =$

$(-1)^{\mathfrak{s}} \cdot \mathfrak{m} \cdot 2^{\rho_{\mathfrak{m}}\mathfrak{e}-\text{bias}}$ *and* $\mathfrak{f}_{\text{out}} = \mathsf{DecryptFloat}_{\boldsymbol{s}}(\mathbf{ct}_{\mathfrak{f}_{\text{out}}}) = (-1)^{\mathfrak{s}_{\text{out}}} \cdot \mathfrak{m}_{\text{out}} \cdot 2^{\rho_{\mathfrak{m}}\mathfrak{e}_{\text{out}}-\text{bias}}$. *If* $\mathfrak{m} \in \left[2^{\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)}, 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}}\right]$, *then* $\mathfrak{m}_{\text{out}} \in \left[2^{\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)}, 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}}\right]$. *Else if* $\mathfrak{m} \in \left[2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}}, 2^{\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}+1)}\right]$, *then* $\mathfrak{m}_{\text{out}} \in \left[2^{\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)}, 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}}\right]$ *and* $\mathfrak{e}_{\text{out}} = \mathfrak{e}+1$. *Moreover, for all* $i \in [0, \ell_{\mathfrak{m}}-1]$ *(resp.* $i \in [0, \ell_{\mathfrak{e}}-1]$*),* $\mathfrak{m}_{\text{out},i} \in [0, 2^{\rho_{\mathfrak{m}}}-1]$ *(resp.* $\mathfrak{e}_{\text{out},i} \in [0, 2^{\rho_{\mathfrak{e}}}-1]$*).*

Proof 5 (Correctness of CarryPropagateFloat (Algorithm 6)) *In Line 2, we get* $\mathbf{ct}_{\mathfrak{e}'} = [\mathsf{ct}_{\mathfrak{e}',\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}',0}]$ *such that* $\forall i \in [0, \ell_{\mathfrak{e}}-1], \mathsf{Decrypt}(\mathsf{ct}_{\mathfrak{e}',i}) = \mathfrak{e}'_i < 2^{\rho_{\mathfrak{e}}}$ *after the carry propagation. Likewise, in Line 3, we do an extended carry propagation of the mantissa, so that* $\mathbf{ct}_{\mathfrak{m}'} = [\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}, \cdots, \mathsf{ct}_{\mathfrak{m}',0}]$ *such that,* $\forall i \in [0, \ell_{\mathfrak{m}}], \mathsf{Decrypt}(\mathsf{ct}_{\mathfrak{m}',i}) = \mathfrak{m}'_i < 2^{\rho_{\mathfrak{m}}}$. *Note that the carry on the most significant block is not lost and create a new* LWE *ciphertext* $\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}$ *encrypting the propagated carry of* $\mathsf{ct}_{\mathfrak{m},\ell_{\mathfrak{m}}-1}$.

In the next steps, the idea is to decide if we need to keep this block $\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}$ *and remove the least significant block (i.e., return* $\mathbf{ct}_{\mathfrak{m}_+}$*) or if we can discard it (i.e., return* $\mathbf{ct}_{\mathfrak{m}_-}$*). This allows us to output a result which has the same number of blocks* $\ell_{\mathfrak{m}}$ *than the input. To do so, in Line 6 we perform a circuit bootstrapping returning a* GGSW *ciphertext:* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(0)$ *if the new* $\mathsf{ct}_{m',\ell_{\mathfrak{m}}}$ *is in* $\mathsf{LWE}_{\boldsymbol{s}}(0)$*, otherwise,* $\overline{\overline{\mathsf{CT}}}$ *is in* $\mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(1)$. *Next in Line 7, Algorithm 5 returns* $\mathbf{ct}_{\mathfrak{m}_+}$ *if* $\overline{\overline{\mathsf{CT}}}$ *is in* $\mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(1)$*, or* $\mathbf{ct}_{\mathfrak{m}_-}$ *otherwise. Likewise, in the case where* $\mathsf{ct}_{\mathfrak{m}',\ell_{\mathfrak{m}}}$ *does not encrypt* 0*, the exponent should be updated. The condition is then the same as previously, so that we can use the same selector* $\overline{\overline{\mathsf{CT}}}$ *to choose between the initial value of the exponent* $\mathbf{ct}_{\mathfrak{e}'}$ *or the one which has been increased by one* $\mathbf{ct}_{\mathfrak{e}'} + \mathsf{TrivialEncrypt}(1, \ell_{\mathfrak{e}})$.

Remark 5 (Carry Propagation & Refresh) *After most operations, we will apply Algorithm 6 to properly propagate the carries and refresh the noise. However, after operations like the ReLU (Algorithm 12) or the approximated Sigmoid (Algorithm 13) that do not fill the carry block, we only need to perform a* PBS *on each ciphertext to obtain fresh noise.*

Lemma 6 (Noise Constraints of Algorithm 6) *The output ciphertexts of Algorithm 6,* $\mathbf{ct}_{\mathfrak{m}_{\text{out}}}$ *has a noise variance* $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{CMux}}$*,* $\mathbf{ct}_{\mathfrak{e}_{\text{out}}}$ *has a noise variance* $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{CMux}}$ *and* $\mathsf{ct}_{\mathfrak{s}_{\text{out}}}$ *has a noise variance* σ^2_{BR}.

To guarantee correctness of Algorithm 6, we need to find parameters that verify the following inequalities:

$$\sigma^2_{\text{in},\mathfrak{e}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2 \quad \& \quad \sigma^2_{\text{in},\mathfrak{m}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2.$$

With $\sigma_{\mathsf{in},\mathfrak{e}}$ the noise variance of the input exponent ciphertexts, $\sigma_{\mathsf{in},\mathfrak{m}}$ the noise variance of the input mantissa ciphertexts and with σ_{BR} the noise added by the blind rotation, σ_{KS} the noise added by the keyswitch, σ_{CMux} the noise added by the CMux *and finally σ_{MS}, the noise added by the modulus switch.*

Proof 6 (Proof of Lemma 6) *In this proof, we use the same techniques as those introduced in [BBB+22]. In particular, we use the noise bound [BBB+22, Def. 8], a quantity representing the maximum noise variance that still guarantees the correctness up to some failure probability. We also use a simpler version of [BBB+22, Theorem 1] which consists of removing redundant inequalities and dominated constraints. Simply put, if we have two inequalities $f(x)+g(x) \leq t$ and $f(x) \leq t$ with f and g two positive functions, we can focus on the first one as the second one will be automatically satisfied if the first is. In this proof and the next ones, when this situation arises, we will say that the second inequality is dominated by the first.*

Let us assume that the input ciphertexts $\mathsf{ct}_{\mathfrak{s}}, \mathbf{ct}_{\mathfrak{e}}$ and $\mathbf{ct}_{\mathfrak{m}}$ have respectively the following noise variances $\sigma^2_{\mathsf{in},\mathfrak{s}}, \sigma^2_{\mathsf{in},\mathfrak{e}}$ and $\sigma^2_{\mathsf{in},\mathfrak{m}}$. The first line of the algorithm consists in a keyswitch and a bootstrapping. We have the following noise constraint: $\sigma^2_{\mathsf{in},\mathfrak{s}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$, with σ^2_{KS} and σ^2_{MS} the noise added respectively by the keyswitch and the modulus switch. t is a noise bound such that $t = \frac{\Delta}{2 \cdot z^*(p_{\mathsf{fail}})}$ with the standard score $z^*(p_{\mathsf{fail}}) = \sqrt{2} \cdot \mathsf{erf}^{-1}(1-p_{\mathsf{fail}})$ and the scaling factor Δ introduced in Definition 1. If we find parameters that guarantee the inequality above, the bootstrapping will be successful with probability $1 - p_{\mathsf{fail}}$.

Then, we have a carry propagate and an extended carry propagate. We refer to the analysis for Algorithm 2 for the explanation about the constraints in these algorithms:

$$\sigma^2_{\mathsf{in},\mathfrak{e}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2 \;\; \& \;\; \sigma^2_{\mathsf{in},\mathfrak{m}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2.$$

Finally, we have a circuit bootstrapping that also creates a noise constraint $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$. Notice that the left-hand side here is smaller than in both inequalities above, so we can remove this last inequality from the set of constraints. Then, the output ciphertexts $\mathbf{ct}_{\mathfrak{m}_{\mathsf{out}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{out}}}$ and $\mathsf{ct}_{\mathfrak{s}_{\mathsf{out}}}$ have respectively a noise variance $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ and σ^2_{BR} with σ^2_{cmux} the variance added by an extended CMux *using a* GGSW *coming from a circuit bootstrapping.*

4 Addition and Subtraction of HFP

In this section, we detail the algorithms used to perform addition and subtraction operations with our floating-point representation. Initially, we describe the operations that manage the mantissa: the first aligns the two mantissas, and the second carries out the subtraction between them, followed by a realignment of the resulting value. Ultimately, the application of these algorithms enables us to efficiently implement homomorphic floating-point (HFP) addition and subtraction operations.

4.1 Managing Mantissas and Exponents

To add two floating-point numbers, we can not directly add their mantissas. First, we need their exponents to be equal and the mantissas to be aligned properly. In what follows, we describe the algorithms to homomorphically perform these operations.

4.1.1 Aligned Mantissa

Algorithm 7 takes as input ciphertexts encrypting two mantissas and their corresponding exponents, and returns the largest exponent $\mathfrak{e}_{\max}$ along with both aligned mantissas. The first step of this operation is to perform a subtraction between the two exponents to obtain $d = |\mathfrak{e}_1 - \mathfrak{e}_2|$ and the sign of this difference. The sign then allows us to select the larger exponent and the mantissa that needs to be aligned. Finally, a tree of CMux, using the bits of d, aligns the selected mantissa by removing the d least significant ciphertexts from the mantissa associated with $\mathfrak{e}_{\min}$. All the steps of this operation are illustrated in Figure 2.

Lemma 7 (Aligned mantissa (Algorithm 7)) *Let* $\mathbf{ct}_{\mathfrak{m}_i}$ *and* $\mathbf{ct}_{\mathfrak{e}_i}$ *such that* $\mathbf{ct}_{\mathfrak{e}_i} = \left[\mathsf{ct}_{\mathfrak{e}_{i,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{i,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{i,\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{i,0}\right)\right]$ *and* $\mathbf{ct}_{\mathfrak{m}_i} = \left[\mathsf{ct}_{\mathfrak{m}_{i,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{i,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{i,\ell_\mathfrak{m}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{i,0}\right)\right]$ *with* $i \in \{1, 2\}$ *be two ciphertexts encrypting* $\mathfrak{m}_i \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_i - \mathsf{bias}}$. *Let* $(\mathbf{ct}_{\mathfrak{m}'_{1_{\mathsf{res}}}}, \mathbf{ct}_{\mathfrak{m}'_{2_{\mathsf{res}}}}, \mathbf{ct}_{\mathfrak{e}_{\max}}) \leftarrow \mathsf{AlignMantissa}(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathbf{ct}_{\mathfrak{m}_2}, \mathsf{PUB})$. *Then,* $\mathfrak{e}_{\max} = \max(\mathfrak{e}_1, \mathfrak{e}_2)$. *If* $\mathfrak{e}_1 > \mathfrak{e}_2$, $\mathfrak{m}'_{1_{\mathsf{res}}} = \mathfrak{m}_1$, *then* $\mathfrak{m}'_{2_{\mathsf{res}}} = \left\lfloor \mathfrak{m}_2 / 2^{\rho_\mathfrak{m} \cdot d} \right\rfloor$ *with* $d = \mathfrak{e}_1 - \mathfrak{e}_2$. *Else if* $\mathfrak{e}_1 < \mathfrak{e}_2$, $\mathfrak{m}'_{2_{\mathsf{res}}} = \mathfrak{m}_2$, *then* $\mathfrak{m}'_{1_{\mathsf{res}}} = \left\lfloor \mathfrak{m}_1 / 2^{\rho_\mathfrak{m} \cdot d} \right\rfloor$ *with* $d = \mathfrak{e}_2 - \mathfrak{e}_1$. *Else if* $\mathfrak{e}_1 = \mathfrak{e}_2$, *then* $\mathfrak{m}'_{1_{\mathsf{res}}} = \mathfrak{m}_1$ *and* $\mathfrak{m}'_{2_{\mathsf{res}}} = \mathfrak{m}_2$. *The complexity of the algorithm is:* $\mathbb{C}_{\mathsf{AlignMantissa}} = (\ell_\mathfrak{e} \cdot \rho_\mathfrak{e} + 1) \cdot \mathbb{C}_{\mathsf{CBS}} + \mathbb{C}^{\ell_\mathfrak{e}}_{\mathsf{IntSub}^*}$

Algorithm 7: $(\mathbf{ct}_{m_{1'}}, \mathbf{ct}_{m_{2'}}, \mathbf{ct}_e) \leftarrow \mathsf{AlignMantissa}(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathbf{ct}_{\mathfrak{m}_2}, \mathsf{PUB})$

Input:
- $\mathbf{ct}_{\mathfrak{e}_1} = [\mathsf{ct}_{\mathfrak{e}_1,\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_1,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1_{\ell_\mathfrak{e}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{m}_1} = [\mathsf{ct}_{\mathfrak{m}_1,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_1,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1_{\ell_\mathfrak{m}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{e}_2} = [\mathsf{ct}_{\mathfrak{e}_2,\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_2,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{2_{\ell_\mathfrak{e}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{2_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{m}_2} = [\mathsf{ct}_{\mathfrak{m}_2,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_2,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{2_{\ell_\mathfrak{m}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{2_0}\right)\right]$
- PUB : Public materials for **PBS-KS** and for CBS

Output:
- $\mathbf{ct}_{\mathfrak{m}'_1} = \left[\mathsf{ct}_{\mathfrak{m}'_1,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}'_1,0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}'_{1_{\ell_\mathfrak{m}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}'_{1_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{m}'_2} = \left[\mathsf{ct}_{\mathfrak{m}'_2,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}'_2,0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}'_{2_{\ell_\mathfrak{m}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}'_{2_0}\right)\right]$
- $\mathbf{ct}_\mathfrak{e} = [\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)]$

```
/* Subtraction between the two exponents follows by the bit
   extraction                                                    */
```
1 $(\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_d = [\mathsf{ct}_{d,\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{d,0}]) \leftarrow \mathsf{IntSub}^*\left(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathsf{PUB}\right)$; /* Algo 2 */

2 **for** *i in [0..$\ell_\mathfrak{e}$-1]* **do**

3 ⎸ **for** *j in [0..$\rho_\mathfrak{e}$-1]* **do**

⎸ ⎸ /* Extract each bit of $\mathbf{ct}_d$ */

4 ⎸ ⎸ $\overline{\overline{\mathsf{CT}}}_{d_{(i\cdot\rho_\mathfrak{e}+j)}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{d,i}, \mathsf{PUB}\right)$

5 $\overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_\mathfrak{s}, \mathsf{PUB}\right)$

/* selects the CT we need to align */

6 $\mathbf{ct}^{\langle 0\rangle}_{\mathfrak{m}_{\mathsf{out}}} = [\mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},0}] \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_2}, \mathbf{ct}_{\mathfrak{m}_1}, \overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}}\right)$; /* Algo 5 */

7 **for** *i in* $[1..\ell_\mathfrak{m}]$ **do**

⎸ /* Remove the i^{th} less significant blocks and add i trivial Zero LWEs on the most significant blocks */

8 ⎸ $\mathbf{ct}^{\langle i\rangle}_{\mathfrak{m}_{\mathsf{out}}} \leftarrow [\mathsf{TrivialEncrypt}\left(0, i\right), \mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},\ell_\mathfrak{m}-1}, ..., \mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},i}]$

9 **for** *i in* $[0..\ell_\mathfrak{e}\cdot\rho_\mathfrak{e})$ **do**

10 ⎸ **if** $\left\lfloor \ell_\mathfrak{m}/2^{i+1} \right\rfloor = 0$ **then**

11 ⎸ ⎸ $\mathbf{ct}^{\langle 0\rangle}_{\mathfrak{m}_{\mathsf{out}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}^{\langle 0\rangle}_{\mathfrak{m}_{\mathsf{out}}}, \mathsf{TrivialEncrypt}\left(0, \ell_\mathfrak{m}\right), \overline{\overline{\mathsf{CT}}}_{d_i}\right)$; /* Algo 5 */

12 ⎸ **else**

13 ⎸ ⎸ **for** *j in* $\left[0..\left\lfloor \ell_\mathfrak{m}/2^{i+1} \right\rfloor\right]$ **do**

⎸ ⎸ ⎸ /* If $\mathbf{ct}^{\langle 2\cdot j+1\rangle}_{\mathfrak{m}_{\mathsf{out}}}$ is not defined, then it is equal to $\mathsf{TrivialEncrypt}\left(0, \ell_\mathfrak{m}\right)$ */

14 ⎸ ⎸ ⎸ $\mathbf{ct}^{\langle i\rangle}_{\mathfrak{m}_{\mathsf{out}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}^{\langle 2\cdot j\rangle}_{\mathfrak{m}_{\mathsf{out}}}, \mathbf{ct}^{\langle 2\cdot j+1\rangle}_{\mathfrak{m}_{\mathsf{out}}}, \overline{\overline{\mathsf{CT}}}_{d_i}\right)$; /* Algo 5 */

15 $\mathbf{ct}_{\mathfrak{m}'_1} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}^{\langle 0\rangle}_{\mathfrak{m}_{\mathsf{out}}}, \overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}}\right)$,
$\mathbf{ct}_{m'_2} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}^{\langle 0\rangle}_{\mathfrak{m}_{\mathsf{out}}}, \mathbf{ct}_{\mathfrak{m}_2}, \overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}}\right)$; /* Algo 5 */

16 $\mathbf{ct}_\mathfrak{e} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}}\right)$; /* Algo 5 */

17 **return** $\left(\mathbf{ct}_{\mathfrak{m}'_1}, \mathbf{ct}_{\mathfrak{m}'_2}, \mathbf{ct}_\mathfrak{e}\right)$

Proof 7 (Correctness of AlignMantissa (Algorithm 7)) *In what follows, we use the index of a ciphertext to refer to the plaintext value it encrypts, i.e.,* $\mathfrak{z} = \mathsf{Decrypt}_s(\mathsf{ct}_\mathfrak{z})$. *From Algorithm 2, we have that* $d = |\mathfrak{e}_1 - \mathfrak{e}_2|$ *and* $\mathfrak{s} = 0$ *if* $\mathfrak{e}_1 \geq \mathfrak{e}_2$, 1 *otherwise. Each bits of* $\mathsf{ct}_\mathfrak{s}$ *and* $\mathbf{ct}_d$ *are converted to* GGSW *via a* CBS, *so that after Line 5, we have:* $\left\{\overline{\overline{\mathsf{CT}}}_{d_i} \in \mathsf{GGSW}_{\boldsymbol{S}}\right\}(d_i)_{i\in[0,\ell_\mathfrak{e}\cdot\rho_\mathfrak{e}-1]}$ *(such that* $d = \sum_{i=0}^{\ell_\mathfrak{e}\cdot\rho_\mathfrak{e}} d_i 2^i$*) and* $\overline{\overline{\mathsf{CT}}}_\mathfrak{s} \in \mathsf{GGSW}_{\boldsymbol{S}}(d_\mathfrak{s})$. *From Algorithm 5, after Line 6, we get* $\mathfrak{m}_{\mathsf{out}}^{\langle 0\rangle} = \mathfrak{m}_2$ *if* $d_\mathfrak{s} = 0$, $\mathfrak{m}_1$ *otherwise. This gives which of the mantissa needs to be aligned, i.e., if the ciphertext* $\overline{\overline{\mathsf{CT}}}_{d_\mathfrak{s}}$ *is in* $\mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(0)$, $\mathfrak{e}_1 \geq \mathfrak{e}_2$ *so we need to align* $\mathfrak{m}_2$, *otherwise* $\mathfrak{e}_1 < \mathfrak{e}_2$, *and thus* $\mathfrak{m}_1$ *needs to be aligned). In next loop, encryption of all possible mantissa shifts are created, i.e.,* $M^{\langle 0\rangle} = \left\{\mathbf{ct}^{\langle i\rangle}_{\mathfrak{m}_{\mathsf{out}}} = [\mathbf{0},\ldots,\mathbf{0},\mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},\ell_\mathfrak{m}-1},\ldots,\mathsf{ct}_{\mathfrak{m}_{\mathsf{out}},i}]\right\}_{i\in[1,\ell_\mathfrak{m}-1]}$ *with* $\mathbf{0} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$, *s.t.* $\mathfrak{m}^{\langle i\rangle}_{\mathsf{out}} = \left\lfloor \mathfrak{m}^{\langle 0\rangle}_{\mathsf{out}}/2^{\rho_\mathfrak{m}\cdot i}\right\rfloor$. *The next steps consists in choosing the right mantissa from this set, depending on the value of* d. *Informally,* d *is the number of blocks by which the mantissa should be shifted. At each step* i *of the loop, the set* $M^{\langle i\rangle}$ *is updated with respect of the binary value of* d, *to contains the encryption of each value in* $\left\{\mathbf{ct}^{\langle\alpha\rangle}_{\mathfrak{m}_{\mathsf{out}}} = \left\lfloor \mathfrak{m}^{\langle 0\rangle}_{\mathsf{out}}/2^{\rho_\mathfrak{m}\cdot\alpha}\right\rfloor \text{ s.t. } \alpha = \sum_{j=0}^{i-1} d_j 2^j \mod 2^{i+1}\right\}$. *Note that in the case where* $\mathfrak{e}_1 = \mathfrak{e}_2$, *the cmux tree will return the selected mantissa without any change. In the end, the set is reduced to a singleton containing* $\left\lfloor \mathfrak{m}^{\langle 0\rangle}_{\mathsf{out}}/2^{\rho_\mathfrak{m}\cdot d}\right\rfloor$. *Finally, the last three* CMux *replace the mantissa value by the aligned one and select the bigger exponent.*

Lemma 8 (Noise Constraints of Algorithm 7) *The output noise variances of the ciphertexts of Algorithm 7,* $\mathbf{ct}_{m'_i}$ *and* $\mathbf{ct}_\mathfrak{e}$, *are respectively* $\sigma^2_{\mathsf{in},m} + \frac{(\rho_e\cdot\ell_e+3)}{2}\cdot\sigma^2_{\mathsf{cmux}}$ *and* $\sigma^2_{\mathsf{in},e} + \sigma^2_{\mathsf{cmux}}$.

To guarantee correctness of Algorithm 7, we need to find parameters that verify the following inequalities:

$$4\cdot\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$$

With $\sigma_{\mathsf{in},\mathfrak{e}}$ *the noise variance of the input exponent ciphertexts,* $\sigma_{\mathsf{in},\mathfrak{m}}$ *the noise variance of the input mantissa ciphertexts and with* σ_{BR} *the noise added by the blind rotation,* σ_{KS} *the noise added by the keyswitch,* σ_{CMux} *the noise added by the* CMux *and finally* σ_{MS}, *the noise added by the modulus switch.*

Proof 8 (Proof of Lemma 8) *Let us assume that the input ciphertexts* $\mathbf{ct}_{\mathfrak{e}_i}$ *and* $\mathbf{ct}_{\mathfrak{m}_i}$ *have respectively a noise variance* $\sigma^2_{\mathsf{in,e}}$ *and* $\sigma^2_{\mathsf{in,m}}$. *The first*

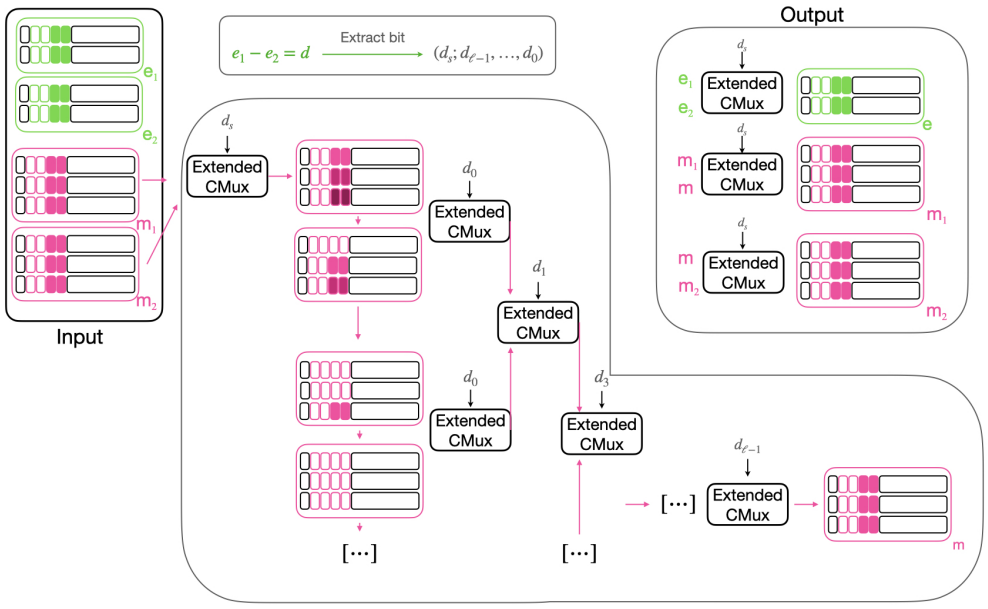

Figure 2: This figure illustrates the main steps of the Homomorphic Align-Mantissa operation (see Algorithm 7). The goal is to align the mantissas based on their exponents. At a high level, we first use Algorithm 2 on the two exponents (in green) to compute the difference between the two exponents, along with the sign. Using the sign, we can determine which mantissa (in pink) is smaller. Then, with the difference and a tree of Extended CMux operations, we can decide how many ciphertexts are needed to increase the smaller mantissa and align the exponents. The final step involves correctly ordering the two mantissas and selecting the larger exponent.

line calls Algorithm 2 (see Lemma 2 for more details). In particular, we compute the output noise of Algorithm 2. The noise variances of ct_s *and* $\mathbf{ct}_{\boldsymbol{d}}$ *are respectively* $4 \cdot \sigma^2_{\mathsf{BR}}$ *and* σ^2_{BR} *with* σ^2_{BR}*, the noise variance of a freshly bootstrapped ciphertext. In the next lines, the algorithm heavily relies on circuit bootstrapping which gives us the following noise constraints :* $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$ & $4 \cdot \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$*, with* σ^2_{KS} *and* σ^2_{MS} *respectively the noise variance added by a keyswitch and by a modulus switch.* t^2 *represents the noise bound as previously defined in the proof of Lemma 6. As the first constraint is dominated by the second, we can remove it from*

the set of constraints. Then, we apply an extended cmux and we create $\ell_{\mathfrak{m}}$ *vectors of ciphertexts composed of outputs of the previous extended cmux and trivial ciphertexts. At this stage, we assume that every ciphertext in these vectors has the same noise variance* $\sigma^2_{\mathsf{in},m} + \sigma^2_{\mathsf{cmux}}$ *with* σ^2_{cmux} *the noise added by a cmux using a GGSW coming from a circuit bootstrap with a noise variance* $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{PPKS}}$. *Then, we apply an extended cmux tree of depth* $\rho_e \cdot \ell_e$. *Therefore, at the end of the tree, the ciphertexts have a noise variance* $\sigma^2_{\mathsf{in},m} + (\rho_e \cdot \ell_e + 1) \cdot \sigma^2_{\mathsf{cmux}}$. *At the end of the algorithm, assuming that* $\mathbf{ct}_{\mathfrak{s}}$ *encrypt* 0 *or* 1 *with probability* $\frac{1}{2}$, *the noise variance of* $\mathbf{ct}_{m'_i}$ *is* $\sigma^2_{\mathsf{in},m} + \frac{(\rho_e \cdot \ell_e + 3)}{2} \cdot \sigma^2_{\mathsf{cmux}}$. *The noise variance of* $\mathbf{ct}_{\mathfrak{e}}$ *is* $\sigma^2_{\mathsf{in},e} + \sigma^2_{\mathsf{cmux}}$.

4.1.2 SubMantissa

SubMantissa performs the subtraction of two mantissas and shifts the result such that the most significant block is not empty (unless the result is zero or too small to be represented). It changes the value of the exponent and the sign consequently. To perform this operation, the two mantissas must to be aligned. Algorithm 8 takes as input the encryption of two aligned mantissas, the exponent and the sign, and returns the encryption of the absolute value of the difference of the mantissas, the exponent and the sign of this subtraction.

Lemma 9 (SubMantissa **(Algorithm 8)**) *Let* $\mathbf{ct}_{\mathfrak{m}_i} = [\mathsf{ct}_{\mathfrak{m}_i,\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_i,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{i_{\ell_{\mathfrak{m}}-1}}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{i_0}\right)\right]$ *with* $i \in \{0,1\}$ *be ciphertexts encrypting* $\mathfrak{m}_i < 2^{\rho_{\mathfrak{m}}}$. *Let* $\mathbf{ct}_{\mathfrak{e}} = [\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, ...\mathsf{ct}_{\mathfrak{e},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)]$ *a ciphertexts encrypting* $\mathfrak{e} < 2^{\rho_{\mathfrak{e}}}$. *Let* $\mathsf{ct}_{\mathfrak{s}_1} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_1)$ *and* $\mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_2)$ *with* $\mathfrak{s}_1 = 1 - \mathfrak{s}_2$ *such that* $\mathfrak{f}_1 = (-1)^{\mathfrak{s}_1} \cdot \mathfrak{m}_1 \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}-\mathsf{bias}}$ *and* $\mathfrak{f}_2 = (-1)^{\mathfrak{s}_2} \cdot \mathfrak{m}_2 \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}-\mathsf{bias}}$. *Let* $\mathbf{ct}_{\mathfrak{f}_{\mathsf{sub}}} = (\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}}) \leftarrow \mathsf{SubMantissa}(\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},1}}, \mathbf{ct}_{\mathfrak{m}_{\mathsf{in},2}}, \mathbf{ct}_{\mathfrak{e}}, \mathsf{ct}_{\mathfrak{s}_1}, \mathsf{PUB})$. *Then* $\mathsf{DecryptFloat}_{\boldsymbol{s}}(\mathbf{ct}_{\mathfrak{f}_{\mathsf{sub}}}) = \mathfrak{f}_1 - \mathfrak{f}_2 = (-1)^{\mathfrak{s}_{\mathsf{sub}}} \cdot \mathfrak{m}_{\mathsf{sub}} \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}_{\mathsf{sub}}-\mathsf{bias}}$ *such that* $\mathfrak{s}_{\mathsf{sub}} = \mathfrak{s}_1$ *if* $\mathfrak{m}_1 \geq \mathfrak{m}_2$, *or* $\mathfrak{s}_2$ *if* $\mathfrak{m}_1 < \mathfrak{m}_2$. *Assuming* $\mathfrak{m}_1 \neq \mathfrak{m}_2$, *let* α *be the index of the first non zero block of* $\mathfrak{m} = |\mathfrak{m}_1 - \mathfrak{m}_2|$ *i.e.,* $\alpha = \min_{i \in [\![0, \ell_{\mathfrak{m}}-1]\!]} \{\ell_{\mathfrak{m}} - 1 - i \text{ s.t. } \mathfrak{m}_i \neq 0\}$ *then, if* $\mathfrak{e} \geq \alpha$ $\mathfrak{e}_{\mathsf{sub}} = \mathfrak{e} - \alpha$ *and* $\mathfrak{m}_{\mathsf{sub}} = |\mathfrak{m}_1 - \mathfrak{m}_2| \cdot 2^{\rho_{\mathfrak{m}} \cdot \alpha}$. *Else if* $\mathfrak{m}_1 = \mathfrak{m}_2$ *or if* $\mathfrak{e} - \alpha < 0$, *then,* $\mathfrak{m}_{\mathsf{sub}} = 0$, $\mathfrak{e}_{\mathsf{sub}} = 0$.
The complexity of the algorithm is: $\mathbb{C}_{\mathsf{SubMantissa}} = \mathbb{C}^{\ell_{\mathfrak{m}}}_{\mathsf{IntSub}^*} + \mathbb{C}^{\ell_{\mathfrak{e}}}_{\mathsf{IntSub}^*} + (\ell_{\mathfrak{m}} + 1) \cdot \mathbb{C}_{\mathsf{CBS}}$

Algorithm 8: $(\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}}) \leftarrow \mathsf{SubMantissa}(\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},1}}, \mathbf{ct}_{\mathfrak{m}_{\mathsf{in},2}}, \mathbf{ct}_{\mathfrak{e}}, \mathsf{ct}_{\mathfrak{s}_1}, \mathsf{PUB})$

Input:
- $\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},1}} = \left[\mathsf{ct}_{\mathfrak{m}_{\mathsf{in},1},\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_{\mathsf{in},1},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{in},1_{\ell_{\mathfrak{m}}-1}}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{in},1_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},2}} = \left[\mathsf{ct}_{\mathfrak{m}_{\mathsf{in},2},\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_{\mathsf{in},2},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{in},2_{\ell_{\mathfrak{m}}-1}}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{in},2_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{e}} = \left[\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, ...\mathsf{ct}_{\mathfrak{e},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)\right]$
- $\mathsf{ct}_{\mathfrak{s}_1} \in \mathsf{LWE}_{\boldsymbol{s}}\left(sign_1\right)$
- PUB : Public keys for KS-PBS and for CBS

Output:
- $\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}} = \left[\mathsf{ct}_{\mathfrak{m}_{\mathsf{sub}},\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_{\mathsf{sub}},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{sub}_{\ell_{\mathfrak{m}}-1}}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\mathsf{sub}_0}\right)\right]$
- $\mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}} = \left[\mathsf{ct}_{\mathfrak{e}_{\mathsf{sub}},\ell_{\mathfrak{e}}-1}, ...\mathsf{ct}_{\mathfrak{e}_{\mathsf{sub}},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\mathsf{sub}_{\ell_{\mathfrak{e}}-1}}\right), ...\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\mathsf{sub}_0}\right)\right]$
- $\mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}_{\mathsf{sub}}\right)$

$(\mathsf{ct}_{\mathfrak{s}}, \mathbf{ct}_{\mathfrak{m}_0} = [\mathsf{ct}_{\mathfrak{m}_0,\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_0,0}]) \leftarrow \mathsf{IntSub}^*\left(\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},1}}, \mathbf{ct}_{\mathfrak{m}_{\mathsf{in},2}}, \mathsf{PUB}\right)$
$\mathbf{ct}_{\mathfrak{e}_0} \leftarrow \mathsf{TrivialEncrypt}\left(0, \ell_{\mathfrak{e}}\right)$
for i *in* $[1..\ell_{\mathfrak{m}}]$ **do**
  $\mathbf{ct}'_{\mathfrak{e}_i} \leftarrow \mathsf{TrivialEncrypt}\left(i, \ell_{\mathfrak{e}}\right)$
  $\mathsf{ct}_0 \leftarrow \mathsf{TrivialEncrypt}\left(0, 1\right)$
  $\mathbf{ct}_{\mathfrak{m}_i} \leftarrow \left[\mathsf{ct}_{\mathfrak{m}_{i-1},\ell_{\mathfrak{m}}-2}, \cdots, \mathsf{ct}_{\mathfrak{m}_{i-1},0}, \mathsf{ct}_0\right]$
  /* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}\left(0\right)$ `if` $\mathsf{ct}_{m_{i-1},\ell_{\mathfrak{m}}-1} \in \mathsf{LWE}_{\boldsymbol{s}}\left(0\right)$, $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}\left(1\right)$ `otherwise` */
  $\overline{\overline{\mathsf{CT}}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{\mathfrak{m}_{i-1},\ell_{\mathfrak{m}}-1}, \mathsf{PUB}\right)$
  $\mathbf{ct}_{\mathfrak{m}_i} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_i}, \mathbf{ct}_{\mathfrak{m}_{i-1}}, \overline{\overline{\mathsf{CT}}}\right)$
  if $i \neq \ell_{\mathfrak{m}}$ **then** $\mathbf{ct}'_{\mathfrak{e}_i} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}'_{\mathfrak{e}_i}, \mathbf{ct}'_{\mathfrak{e}_{i-1}}, \overline{\overline{\mathsf{CT}}}\right)$ **else** $\mathbf{ct}'_{\mathfrak{e}_i} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}}, \mathbf{ct}'_{\mathfrak{e}_{i-1}}, \overline{\overline{\mathsf{CT}}}\right)$;
$(\mathsf{ct}_{\mathfrak{s}_{\mathfrak{e}}}, \mathbf{ct}_{\mathfrak{e}_{res}}) \leftarrow \mathsf{IntSub}^*\left(\mathbf{ct}_{\mathfrak{e}}, \mathbf{ct}'_{\mathfrak{e}_{\ell_{\mathfrak{m}}}}, \mathsf{PUB}\right)$; /* Algo 2 */
/* $\overline{\overline{\mathsf{CT}}}_{s_{\mathfrak{e}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}\left(0\right)$ `if` $\mathsf{ct}_{\mathfrak{s}_{\mathfrak{e}}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(0\right)$, $\overline{\overline{\mathsf{CT}}}_{\mathfrak{s}_{\mathfrak{e}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}\left(1\right)$ `otherwise` */
$\overline{\overline{\mathsf{CT}}}_{\mathfrak{s}_{\mathfrak{e}}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{\mathfrak{s}_{\mathfrak{e}}}, \mathsf{PUB}\right)$
$\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathsf{TrivialEncrypt}\left(0, \ell_{\mathfrak{m}}\right), \mathbf{ct}_{\mathfrak{m}_{\ell_{\mathfrak{m}}}}, \overline{\overline{\mathsf{CT}}}_{\mathfrak{s}_{\mathfrak{e}}}\right)$,
$\mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathsf{TrivialEncrypt}\left(0, \ell_{\mathfrak{e}}\right), \mathbf{ct}_{\mathfrak{e}_{res}}, \overline{\overline{\mathsf{CT}}}_{\mathfrak{s}_{\mathfrak{e}}}\right)$
$\mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}} \leftarrow \mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}}$
return $(\mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}; \mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}; \mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}})$

Proof 9 (Correctness of SubMantissa (Algorithm 8)) *The first step of the algorithm is to subtract the two mantissas. We obtain* $\mathbf{ct}_{\mathfrak{m}_0}$ *which is equals to* $|\mathbf{ct}_{\mathfrak{m}_{\mathsf{in},1}} - \mathbf{ct}_{\mathfrak{m}_{\mathsf{in},2}}|$ *and* $\mathsf{ct}_{\mathfrak{s}}$ *the sign of this subtraction. As the two mantissas are aligned, we have* $\mathfrak{m}_0$ *in* $[0, 2^{\ell_{\mathfrak{m}} \cdot \rho_{\mathfrak{m}}})$.

At each step i of the loop, we take the previously computed ciphertext $\mathsf{ct}_{\mathfrak{m}_{i-1}}$ encrypting a message $\mathfrak{m}_{i-1}$ and build another ciphertext $\mathsf{ct}_{\mathfrak{m}_i}$ encrypting the message $\mathfrak{m}_i = \mathfrak{m}_{i-1} \cdot 2^{\rho_\mathfrak{m}}$. On line 7, we create a GGSW *ciphertext $\overline{\overline{\mathsf{CT}}}$ encrypting 0 if the most significant block of the mantissa $\mathfrak{m}_{i-1,\ell_\mathfrak{m}-1}$ is equal to 0 and 1 if it contains some non-zero integer. Remember, our goal is to realign the mantissa to stay in the classical representation (i.e., $\mathfrak{m}_{\mathsf{sub}} \in \left[2^{(\ell_\mathfrak{m}-1)\cdot\rho_\mathfrak{m}}, 2^{\ell_\mathfrak{m}\cdot\rho_\mathfrak{m}}\right)$ or $\mathfrak{m}_{\mathsf{sub}} = 0$ if $\mathfrak{m}_{\mathsf{in},1} = \mathfrak{m}_{\mathsf{in},2}$). Therefore, we use a cmux to select $\mathsf{ct}_{\mathfrak{m}_{i-1}}$ if $\mathfrak{m}_{i-1,\ell_\mathfrak{m}-1} \neq 0$ and $\mathsf{ct}_{\mathfrak{m}_i}$ if $\mathfrak{m}_{i-1,\ell_\mathfrak{m}-1} = 0$. In the same way, we use the cmux to update the value of the exponent. To do so, we take the previously computed $\mathsf{ct}_{\mathfrak{e}_{i-1}}$ and create a new ciphertext $\mathsf{ct}_{\mathfrak{e}_i}$, a trivial encryption of i. As we want to select $\mathsf{ct}_{\mathfrak{e}_{i-1}}$ (respectively $\mathsf{ct}_{\mathfrak{e}_i}$) if we selected $\mathsf{ct}_{\mathfrak{m}_{i-1}}$ (resp. $\mathsf{ct}_{\mathfrak{m}_i}$) in the previous step, we can perform a cmux with the same GGSW ciphertext $\overline{\overline{\mathsf{CT}}}$. Assuming that we have $\mathfrak{m}_0 \neq 0$, at the end of the for-loop, $\mathsf{ct}_{\mathfrak{e}_i}$ is encrypting a value α such that $\mathfrak{m}_0 \cdot 2^{\rho_\mathfrak{m}\cdot\alpha} \in \left[2^{(\ell_\mathfrak{m}-1)\cdot\rho_\mathfrak{m}}, 2^{\ell_\mathfrak{m}\cdot\rho_\mathfrak{m}}\right)$ and $\mathsf{ct}_{\mathfrak{m}_i}$ is encrypting $\mathfrak{m}_0 \cdot 2^{\rho_\mathfrak{m}\cdot\alpha}$. If we have $\mathfrak{m}_0 = 0$, $\mathsf{ct}_{\mathfrak{m}i}$ will still be equal to 0.*

The next step is to update the exponent. In line 13, we subtract the value α encrypted in $\mathsf{ct}_{\mathfrak{e}_{\ell_\mathfrak{m}}}$ to $\mathsf{ct}_\mathfrak{e}$. The sign of this subtraction is in $\mathsf{LWE}_{\boldsymbol{s}}(0)$ if we can do the subtraction ($\mathfrak{e} > \alpha$) otherwise, the result is in $\mathsf{LWE}_{\boldsymbol{s}}(1)$, the value of the subtraction is too small to be represented with our encoding. With this sign we create a new GGSW *ciphertext $\overline{\overline{\mathsf{CT}}}$. Finally, the last* CMux *returns the ciphertexts encrypting mantissa $\mathfrak{m}_{\mathsf{sub}} = \mathfrak{m}_0 \cdot 2^{\rho_\mathfrak{m}\cdot\alpha}$ and the ciphertexts encrypting the exponent $\mathfrak{e}_{\mathsf{sub}} = \mathfrak{e} - \alpha$ if the subtraction can be done, otherwise it returns the ciphertexts encrypting 0.*

Lemma 10 (Noise Constraints of Algorithm 8) *The output noise variances of ciphertexts of Algorithm 8, $\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}$ and $\mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}}$, are respectively $\sigma^2_{\mathsf{BR}} + (\ell_\mathfrak{m}+1)\cdot\sigma^2_{\mathsf{cmux}}$, $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ and $\sigma^2_{\mathsf{in},s} + 4\sigma^2_{\mathsf{BR}}$.*

To guarantee correctness of Algorithm 8, we need to find parameters that verify the following inequalities:

$$\sigma^2_{\mathsf{BR}} + (i-1)\cdot\sigma^2_{\mathsf{cmux}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2 \;\;\&\;\; 4\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$$

With $\sigma_{\mathsf{in},\mathfrak{s}}$ the noise variance of the input sign ciphertext and with σ_{BR} the noise added by the blind rotation, σ_{KS} the noise added by the keyswitch, σ_{CMux} the noise added by the CMux *and finally σ_{MS}, the noise added by the modulus switch.*

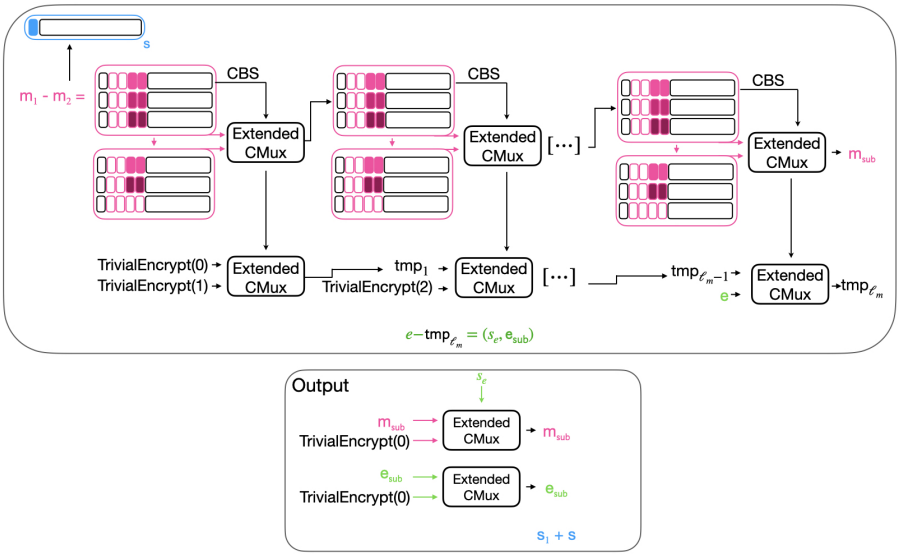

Figure 3: This figure represents the main steps of Algorithm 8. At a high level, we first subtract the two mantissas using Algorithm 2, obtaining the absolute value of the subtraction along with the sign. Next, we perform a loop where, at each step, the first ciphertext of the new mantissa is transformed into a GGSW ciphertext using a CBS. Using this GGSW ciphertext, we remove the most significant ciphertext if it is empty otherwise, we keep the mantissa. At the same time, at each step, we count the number of ciphertexts removed. When the loop finishes, we subtract the number of removed ciphertexts from the exponent. Finally, we use an extended Cmux to return 0 if the exponent is negative or if the mantissa is empty.

Proof 10 (Proof of Lemma 10) *Let us assume that the input ciphertexts* $\mathbf{ct}_{\mathfrak{e}}$, $\mathbf{ct}_{\mathfrak{m}_i}$ *and* $\mathsf{ct}_{\mathfrak{s}_1}$ *have respectively a noise variance* $\sigma^2_{\mathsf{in},\mathfrak{e}}$, $\sigma^2_{\mathsf{in},\mathfrak{m}}$ *and* $\sigma^2_{\mathsf{in},\mathfrak{s}}$.

Using the noise analysis of Algorithm 2 presented in Lemma 2, we know that the noise variance of $\mathbf{ct}_{m_0}$ *and* $\mathsf{ct}_{\mathfrak{s}}$ *are respectively* σ^2_{BR} *and* $4\cdot\sigma^2_{\mathsf{BR}}$. *In the for-loop, for each index* $1 \leq i \leq \ell_{\mathfrak{m}}$, *we can compute the noise constraint* $\sigma^2_{\mathsf{BR}} + (i-1)\cdot\sigma^2_{\mathsf{cmux}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$ *and the noise variance of* $\mathbf{ct}_{\mathfrak{m}_i}$ *is* $\sigma^2_{\mathsf{BR}} + i\cdot\sigma^2_{\mathsf{cmux}}$, *the noise variance of* $\mathbf{ct}_{\mathfrak{e}_i}$ *is* $i\cdot\sigma^2_{\mathsf{cmux}}$ *for* $i \neq \ell_{\mathfrak{m}}$ *and* $\max\left((\ell_{\mathfrak{m}}-1)\cdot\sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{in},e}\right) + \sigma^2_{\mathsf{cmux}}$ *for* $i = \ell_{\mathfrak{m}}$. *As in the previous proofs, we only need to retain the noise constraint for* $i = \ell_{\mathfrak{m}}$, *as it dominates the other constraints. Then, we*

have a call to Algorithm 2 which gives us ciphertexts of variances respectively $4\sigma^2_{\mathsf{BR}}$ *and* σ^2_{BR}. *Next, we perform a circuit bootstrapping which gives us the following constraint* $4\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$. *Finally, the algorithm outputs* $\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}$ *and* $\mathsf{ct}_{\mathfrak{s}_{\mathsf{sub}}}$ *of respective variances* $\sigma^2_{\mathsf{BR}} + (\ell_{\mathfrak{m}}+1)\cdot\sigma^2_{\mathsf{cmux}}$, $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ *and* $\sigma^2_{\mathsf{in},s} + 4\sigma^2_{\mathsf{BR}}$.

4.2 Addition and Subtraction

This operation performs the addition of two homomorphic floating-point numbers. To perform a subtraction, we only need to change the input sign of the second ciphertext. This operation is straightforward, as the sign is on a padding bit, adding the clear integer $q/2$ to the sign ciphertext change the sign of the floating-point.

This operation is based on the previous algorithm. We first need to align the mantissas (Algorithm 7). Next, we perform both the addition and the subtraction (Algorithm 8) of the mantissas and then we choose which of the two results to output based on the signs. All the steps of this operation are illustrated in Figure 4.

After this operation, we need to perform the operation CarryPropagate-Float (Algorithm 6) to retrieve the homomorphic floating-point representation.

Lemma 11 (Addition (Algorithm 9)) *Let* $\mathbf{ct}_{\mathfrak{f}_i}$ *such that* $\mathsf{ct}_{\mathfrak{s}_i} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_i)$,$\mathbf{ct}_{\mathfrak{e}_i} = \left[\mathsf{ct}_{\mathfrak{e}_{i,\ell_{\mathfrak{e}}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{i,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{i,\ell_{\mathfrak{e}}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{i,0})]$ *encrypting* $\mathfrak{e}_i < 2^{\rho_{\mathfrak{e}}}$ *and* $\mathbf{ct}_{\mathfrak{m}_i} = \left[\mathsf{ct}_{\mathfrak{m}_{i,\ell_{\mathfrak{m}}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{i,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,\ell_{\mathfrak{m}}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,0})]$ *encrypting* $\mathfrak{m}_i < 2^{\rho_{\mathfrak{m}}}$ *with* $i \in \{1,2\}$ *be two ciphertexts encrypting* $\mathfrak{f}_i = (-1)^{\mathfrak{s}_i} \cdot \mathfrak{m}_i \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}_i - \mathsf{bias}}$. *Let* $(\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{res}}}) = \mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}} \leftarrow \mathsf{Addition}(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB})$. *Then* $\mathsf{DecryptFloat}(\mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}}) = (-1)^{\mathfrak{s}_{\mathsf{res}}} \cdot \mathfrak{m}_{\mathsf{res}} \cdot (2^{\rho_{\mathfrak{m}}})^{\mathfrak{e}_{\mathsf{res}} - \mathsf{bias}} = \mathfrak{f}_{\mathsf{res}} = \mathfrak{f}_1 + \mathfrak{f}_2 + \epsilon$ *such that if* $\mathfrak{e}_1 \geq \mathfrak{e}_2$, *then* $\mathfrak{m}'_1 = \mathfrak{m}_1$ *and* $\mathfrak{m}'_2 = \lfloor \mathfrak{m}_2 / 2^{\rho_{\mathfrak{m}} \cdot \gamma} \rfloor$ *with* $\gamma = \mathfrak{e}_1 - \mathfrak{e}_2$. *Else if* $\mathfrak{e}_1 \leq \mathfrak{e}_2$, *then* $\mathfrak{m}'_2 = \mathfrak{m}_2$ *and* $\mathfrak{m}'_1 = \lfloor \mathfrak{m}_1 / 2^{\rho_{\mathfrak{m}} \cdot \gamma} \rfloor$ *with* $\gamma = \mathfrak{e}_2 - \mathfrak{e}_1$. *For* ϵ, *we refer to Definition 5. If* $\mathfrak{s}_1 = \mathfrak{s}_2$, *then* $\mathfrak{s}_{\mathsf{res}} = \mathfrak{s}_1$, $\mathfrak{e}_{\mathsf{res}} = \max(\mathfrak{e}_1, \mathfrak{e}_2)$ *and* $\mathfrak{m}_{res} = \mathfrak{m}'_1 + \mathfrak{m}'_2$. *Else if* $\mathfrak{s}_1 \neq \mathfrak{s}_2$ *and if* $\mathfrak{m}'_1 \geq \mathfrak{m}'_2$, *then* $\mathfrak{s}_{\mathsf{res}} = \mathfrak{s}_1$; *or if* $\mathfrak{m}'_1 < \mathfrak{m}'_2$, *then* $\mathfrak{s}_{\mathsf{res}} = \mathfrak{s}_2$. *Assuming* $\mathfrak{m}'_1 \neq \mathfrak{m}'_2$ *and* $\mathfrak{s}_1 \neq \mathfrak{s}_2$, *let* α *be the index of the first non zero block of* $\mathfrak{m} = |\mathfrak{m}'_1 - \mathfrak{m}'_2|$, *if* $\max(\mathfrak{e}_1, \mathfrak{e}_2) \geq \alpha$, *then* $\mathfrak{e}_{\mathsf{res}} = \max(\mathfrak{e}_1, \mathfrak{e}_2) - \alpha$ *and* $\mathfrak{m}_{\mathsf{res}} = |\mathfrak{m}'_1 - \mathfrak{m}'_2| \cdot 2^{\ell_{\mathfrak{m}} \cdot \alpha}$. *Else if* $\max(\mathfrak{e}_1, \mathfrak{e}_2) < \alpha$, *then* $\mathfrak{e}_{\mathsf{res}} = 0$, $\mathfrak{m}_{\mathsf{res}} = 0$ *and* $\mathfrak{e}_{\mathsf{res}} = 0$. *The complexity of the algorithm is:* $\mathbb{C}_{\mathsf{Addition}} = \mathbb{C}_{\mathsf{SubMantissa}} + \mathbb{C}_{\mathsf{AlignMantissa}} + \mathbb{C}_{\mathsf{CBS}}$.

Algorithm 9: $\mathbf{ct}_{\mathfrak{f}_{\text{res}}} \leftarrow \mathsf{Addition}\left(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB}\right)$

Input:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}_1} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_1} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}_1\right) \\ \mathbf{ct}_{\mathfrak{e}_1} = \left[\mathsf{ct}_{\mathfrak{e}_{1,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{1,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1,\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1,0}\right)\right] \\ \mathbf{ct}_{\mathfrak{m}_1} = \left[\mathsf{ct}_{\mathfrak{m}_{1,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{1,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1,\ell_\mathfrak{m}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1,0}\right)\right] \end{cases} \\ \mathbf{ct}_{\mathfrak{f}_2} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}_2\right) \\ \mathbf{ct}_{\mathfrak{e}_2} = \left[\mathsf{ct}_{\mathfrak{e}_{2,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{2,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{2,\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{2,0}\right)\right] \\ \mathbf{ct}_{\mathfrak{m}_2} = \left[\mathsf{ct}_{\mathfrak{m}_{2,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{2,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{2,\ell_\mathfrak{m}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{2,0}\right)\right] \end{cases} \\ \mathsf{PUB} : \text{Public materials for } \textbf{PBS-KS} \text{ and for } \mathsf{CBS} \end{cases}$$

Output:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}_{\text{res}}} = \begin{cases} \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}_{\text{res}}\right) \\ \mathbf{ct}_{\mathfrak{e}_{\text{res}}} = \left[ct_{\mathfrak{e}_{\text{res}}}, \cdots, ct_{\mathfrak{e}_0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\text{res},\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\text{res},0}\right)\right] \\ \mathbf{ct}_{\mathfrak{m}_{\text{res}}} = \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\text{res},\ell_\mathfrak{m}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\text{res},0}\right)\right] \end{cases} \end{cases}$$

1 $\left(\mathbf{ct}_{\mathfrak{m}'_1}, \mathbf{ct}_{\mathfrak{m}'_2}, \mathbf{ct}_{\mathfrak{e}}\right) \leftarrow \mathsf{AlignMantissa}(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathbf{ct}_{\mathfrak{m}_2}, \mathsf{PUB})$; /* Algo 7 */

2 $\mathbf{ct}_{\mathfrak{m}_{\text{add}}} \leftarrow \mathsf{ct}_{\mathfrak{m}'_1} + \mathsf{ct}_{\mathfrak{m}'_2}$

3 $(\mathbf{ct}_{\mathfrak{m}_{\text{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\text{sub}}}, \mathsf{ct}_{\mathfrak{s}_{\text{sub}}}) \leftarrow \mathsf{SubMantissa}(\mathbf{ct}_{\mathfrak{m}'_1}, \mathbf{ct}_{\mathfrak{m}'_2}, \mathbf{ct}_{\mathfrak{e}}, \mathsf{ct}_{\mathfrak{s}_1}, \mathsf{PUB})$; /* Algo 8 */
/* $\overline{\overline{\mathsf{CT}_\mathfrak{s}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(0)$ if $\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$; $\overline{\overline{\mathsf{CT}_\mathfrak{s}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(1)$ otherwise. */

4 $\overline{\overline{\mathsf{CT}_\mathfrak{s}}} = \mathsf{CBS}\left(\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2}, \mathsf{PUB}\right)$

5 $\mathbf{ct}_{\mathfrak{e}_{\text{res}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}}, \mathbf{ct}_{\mathfrak{e}_{\text{sub}}}, \overline{\overline{\mathsf{CT}_\mathfrak{s}}}\right)$,
$\mathbf{ct}_{\mathfrak{m}_{\text{res}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_{\text{add}}}, \mathbf{ct}_{\mathfrak{m}_{\text{sub}}}, \overline{\overline{\mathsf{CT}_\mathfrak{s}}}\right)$; /* Algo 5 */

6 $\mathsf{ct}_{\mathfrak{s}_{\text{res}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathsf{ct}_{\mathfrak{s}_1}, \mathsf{ct}_{\mathfrak{s}_{\text{sub}}}, \overline{\overline{\mathsf{CT}_\mathfrak{s}}}\right)$; /* Algo 5 */

7 **return** $\mathbf{ct}_{\mathfrak{f}_{\text{res}}} = (\mathsf{ct}_{\mathfrak{s}_{\text{res}}}, \mathbf{ct}_{e_{\text{res}}}, \mathbf{ct}_{m_{\text{res}}})$

Proof 11 (Correctness of Addition (Algorithm 9)) *As defined in AlignMantissa (Algorithm 7), line 1 returns* $\mathfrak{e} = \max(\mathfrak{e}_1, \mathfrak{e}_2)$ *and* $\mathsf{ct}_{\mathfrak{m}'_1}$ *and* $\mathsf{ct}_{\mathfrak{m}'_2}$ *aligned (such that if* $\mathfrak{e}_1 \geq \mathfrak{e}_2$, $\mathfrak{m}'_1 = \mathfrak{m}_1$ *and* $\mathfrak{m}'_2 = \lfloor \mathfrak{m}_2 / 2^{\rho_\mathfrak{m} \cdot \gamma} \rfloor$ *with* $\gamma = \mathfrak{e}_1 - \mathfrak{e}_2$. *And if* $\mathfrak{e}_1 \leq \mathfrak{e}_2$, $\mathfrak{m}'_2 = \mathfrak{m}_2$ *and* $\mathfrak{m}'_1 = \lfloor \mathfrak{m}_1 / 2^{\rho_\mathfrak{m} \cdot \gamma} \rfloor$ *with* $\gamma = \mathfrak{e}_2 - \mathfrak{e}_1$.*).*

Line 2 adds the two aligned mantissas $\mathfrak{m}'_1$ *and* $\mathfrak{m}'_2$ *by adding together each* LWE *ciphertext. As the carry block is empty, these operations can be done directly.*

As defined in Algorithm 8, line 3 returns $\mathsf{ct}_{\mathfrak{m}_{\text{sub}}} = |\mathfrak{m}'_1 - \mathfrak{m}'_2| \cdot 2^{\ell_\mathfrak{m} \cdot \alpha}$, $\mathfrak{s}_{\text{sub}} = \mathfrak{s}_1$ *if* $\mathfrak{m}'_1 > \mathfrak{m}'_2$, *or* $\mathfrak{s}_{\text{sub}} = \mathfrak{s}_2$ *if* $\mathfrak{m}'_1 < \mathfrak{m}'_2$. *Assuming* $\mathfrak{m}'_1 \neq \mathfrak{m}'_2$, *let* α *be the index of the first non zero block of* $\mathfrak{m}_{\text{res}} = |\mathfrak{m}'_1 - \mathfrak{m}'_2| \cdot 2^{\ell_\mathfrak{m} \cdot \alpha}$, *then if* $\mathfrak{e} > \alpha$ *the algorithm returns* $\mathfrak{e}_{\text{sub}} = \mathfrak{e} - \alpha$. *If* $\mathfrak{m}'_1 = \mathfrak{m}'_2$ *or* $\mathfrak{e} < \alpha$, $\mathfrak{m}_{sub} = 0$, $\mathfrak{e}_{\text{sub}} = 0$ *and* $\mathfrak{s}_{\text{sub}} = \mathfrak{s}_1$.

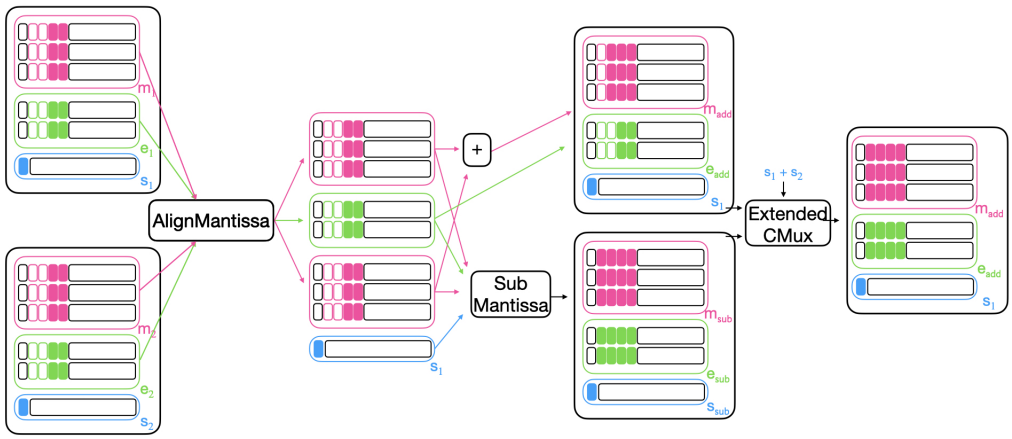

Figure 4: This figure gives an overview of each step needed to perform the homomorphic floating-point addition.

By adding the sign on line 4 and performing a CBS *over the result, we obtain a* GGSW *ciphertext which encrypt* $\mathfrak{s}_1 + \mathfrak{s}_2$. *So* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(0)$ *if the two signs are equal or* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(1)$ *if the signs are different.*

So with this GGSW *ciphertext, we will return the mantissa, the exponent and the sign corresponding to the addition if the signs are equal and if they are different, we return the mantissa, the exponent and the sign corresponding to the subtraction as proposed on the lemma 11.*

Lemma 12 (Noise Constraints of Algorithm 9) *The output noise variances of ciphertexts of Algorithm 9,* $\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}$ *and* $\mathbf{ct}_{\mathfrak{s}_{\mathsf{res}}}$, *are respectively* $\max\left(2\sigma_{\mathsf{BR}}^2 + (\rho_e \ell_e + 5) \cdot \sigma_{\mathsf{cmux}}^2, \sigma_{\mathsf{BR}}^2 + (\ell_m + 1) \cdot \sigma_{\mathsf{cmux}}^2\right) + \sigma_{\mathsf{cmux}}^2, \sigma_{\mathsf{BR}}^2 + 2\sigma_{\mathsf{cmux}}^2$ *and* $5\sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{cmux}}^2$.

To guarantee correctness of this operation, we need to find parameters that verify the following inequalities:

$$2\max\left((\ell_m - 1) \cdot \sigma_{\mathsf{cmux}}^2, \sigma_{\mathsf{BR}}^2 + 2 \cdot \sigma_{\mathsf{cmux}}^2\right) + \sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2$$

$$\max\left(2\sigma_{\mathsf{BR}}^2 + (\rho_e \ell_e + 5)\,\sigma_{\mathsf{cmux}}^2, \sigma_{\mathsf{BR}}^2 + (\ell_m + 1)\,\sigma_{\mathsf{cmux}}^2\right) + \sigma_{\mathsf{cmux}}^2 + \sigma_{\mathsf{BR}}^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2 \leq t^2.$$

With $\sigma_{\mathsf{in},\mathfrak{e}}$ the noise variance of the input exponent ciphertexts, $\sigma_{\mathsf{in},\mathfrak{m}}$ the noise variance of the input mantissa ciphertexts and with σ_{BR} the noise added by the blind rotation, σ_{KS} the noise added by the keyswitch, σ_{cmux} the noise added by the CMux *and finally σ_{MS}, the noise added by the modulus switch.*

Proof 12 (Proof of Lemma 12) *Let us assume that the inputs of this algorithm are the outputs of Algorithm 6. It means that the variances of $\mathbf{ct}_{\mathfrak{s}_i}, \mathbf{ct}_{\mathfrak{e}_i}$ and $\mathbf{ct}_{\mathfrak{m}_i}$ are respectively $\sigma^2_{\mathsf{BR}}, \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ and $\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ (see Lemma 6).*

The first line calls Algorithm 7, we use Lemma 8 to estimate the noise variances of $\mathbf{ct}_{\mathfrak{m}'_1}, \mathbf{ct}_{\mathfrak{m}'_2}$ which are equal to $\sigma^2_{\mathsf{BR}} + \left(\frac{\rho_e \ell_e + 5}{2}\right) \cdot \sigma^2_{\mathsf{cmux}}$ and the noise variance of $\mathbf{ct}_{\mathfrak{e}'}$ which is equal to $\sigma^2_{\mathsf{BR}} + 2 \cdot \sigma^2_{\mathsf{cmux}}$. Then, $\mathbf{ct}_{\mathfrak{m}'_1}$ and $\mathbf{ct}_{\mathfrak{m}'_2}$ are added which doubles the variance.

Next, we call Algorithm 8 and using Lemma 10, we deduce that the noise variances of $\mathbf{ct}_{\mathfrak{m}_{\mathsf{sub}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{sub}}}$ and $\mathbf{ct}_{\mathfrak{s}_{\mathsf{sub}}}$ are respectively $\sigma^2_{\mathsf{BR}} + (\ell_m + 1) \cdot \sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$ and $5\sigma^2_{\mathsf{BR}}$.

Then, we have a circuit bootstrap which must satisfies the following constraint $2\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$. Finally, we have an extended cmux for the mantissa, the exponent and the sign. The noise variances of $\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}$ and $\mathbf{ct}_{\mathfrak{s}_{\mathsf{res}}}$ are respectively $\max\left(2\sigma^2_{\mathsf{BR}} + (\rho_e \ell_e + 5) \cdot \sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + (\ell_m + 1) \cdot \sigma^2_{\mathsf{cmux}}\right) + \sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + 2\sigma^2_{\mathsf{cmux}}$ and $5\sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{cmux}}$.

Using Lemmas 6, 2, 8 and 10 and by noticing that some of the inequalities are dominated by others, we find the complete set of constraints. As we want to be able to chain several additions, we will assume that after the addition, we apply Algorithm 6. The non-dominated set of constraints is the following:

$$2\max\left((\ell_m - 1) \cdot \sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + 2 \cdot \sigma^2_{\mathsf{cmux}}\right) + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2$$
$$\max\left(2\sigma^2_{\mathsf{BR}} + (\rho_e \ell_e + 5)\,\sigma^2_{\mathsf{cmux}}, \sigma^2_{\mathsf{BR}} + (\ell_m + 1)\,\sigma^2_{\mathsf{cmux}}\right) + \sigma^2_{\mathsf{cmux}} + \sigma^2_{\mathsf{BR}} + \sigma^2_{\mathsf{KS}} + \sigma^2_{\mathsf{MS}} \leq t^2.$$

We need to find parameters that verify these inequalities to guarantee correctness.

5 Multiplication and Division

In this section, we introduce a very efficient floating-point multiplication with the HFP representation. Then, we detail a second algorithm to perform division. Finally, we briefly describe how to perform the ReLU and an approximated Sigmoid.

5.1 Multiplication

This operation computes the product of two HFPs. Following this procedure, it is necessary to apply the CarryPropagateFloat algorithm (Algorithm 6). This step ensures that the homomorphic floating-point number is returned to its standard representation, aligning it with the conventional HFP formalism. At a high level, the goal is to multiply the two mantissas without losing precision. Next, the exponent is updated by computing the sum of the two exponents and subtracting the bias. We note that we selected this bias to allow for efficient computation of this step. Due to the representation, we know that if the most significant ciphertext of the mantissa equals zero, or if the exponent is negative, the result of the operation is zero since we do not work with subnormal values. Otherwise, we return the result of the multiplication along with the updated exponent.

Lemma 13 (Multiplication (Algorithm 10)) *Let* $\mathbf{ct}_{\mathfrak{f}_i}$ *such that* $\mathsf{ct}_{\mathfrak{s}_i} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_i)$, $\mathbf{ct}_{\mathfrak{e}_i} = \left[\mathsf{ct}_{\mathfrak{e}_{i,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{i,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{i,\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{i,0})]$ *encrypting* $\mathfrak{e}_i < 2^{\rho_\mathfrak{e}}$ *and* $\mathbf{ct}_{\mathfrak{m}_i} = \left[\mathsf{ct}_{\mathfrak{m}_{i,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{i,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,\ell_\mathfrak{m}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,0})]$ *encrypting* $\mathfrak{m}_i < 2^{\rho_\mathfrak{m}}$ *with* $i \in \{1,2\}$ *be two ciphertexts encrypting* $\mathfrak{f}_i = (-1)^{\mathfrak{s}_i} \cdot \mathfrak{m}_i \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_i - \mathsf{bias}}$. *Let* $(\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{res}}}) = \mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}} \leftarrow \mathsf{Multiplication}(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB})$.

Then $\mathsf{DecryptFloat}(\mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}}) = \mathfrak{f}_{\mathsf{res}} = (-1)^{\mathfrak{s}_{\mathsf{res}}} \cdot \mathfrak{m}_{\mathsf{res}} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_{\mathsf{res}} - \mathsf{bias}} = \mathfrak{f}_1 \cdot \mathfrak{f}_2 + \epsilon$ *such that* $\mathsf{ct}_{\mathfrak{s}_{\mathsf{res}}} = \mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2}$ *and* ϵ *is the maximum error added by the operation as express in Definition 5. If* $\mathfrak{m}_1 \neq 0$ *and* $\mathfrak{m}_2 \neq 0$, *if* $\mathfrak{e}_1 + \mathfrak{e}_2 \geq \mathsf{bias} - \ell_\mathfrak{m} + 1$ *then* $\mathfrak{m}_{\mathsf{res}} = \left\lfloor \mathfrak{m}_1 \cdot \mathfrak{m}_2 / 2^{(\ell_\mathfrak{m}-1)\cdot\rho_\mathfrak{m}} \right\rfloor$ *and* $\mathfrak{e}_{\mathsf{res}} = \mathfrak{e}_1 + \mathfrak{e}_2 - \mathsf{bias} + \ell_\mathfrak{m} - 1$ *with* $|\epsilon| < (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_{\mathsf{res}} - \mathsf{bias}}$. *If* $\mathfrak{e}_1 + \mathfrak{e}_2 < \mathsf{bias} - \ell_\mathfrak{m} + 1$ *then* $\mathfrak{m}_{\mathsf{res}} = 0$ *and* $\mathfrak{e}_{\mathsf{res}} = 0$ *with* $|\epsilon| < 2^{\rho_\mathfrak{m}\cdot(\ell_\mathfrak{m}-1)}(2^{\rho_\mathfrak{m}})^{-\mathsf{bias}}$. *If* $\mathfrak{m}_1 = 0$ *or* $\mathfrak{m}_2 = 0$ *then* $\mathfrak{m}_{\mathsf{res}} = 0$ *and* $\mathfrak{e}_{\mathsf{res}} = 0$ *with* $|\epsilon| < 2^{\rho_\mathfrak{m}\cdot(\ell_\mathfrak{m}-1)}(2^{\rho_\mathfrak{m}})^{-\mathsf{bias}}$.

The complexity of the algorithm is: $\mathbb{C}_{\mathsf{Multiplication}} = \mathbb{C}^{\ell_\mathfrak{m}}_{\mathsf{IntMul}} + 2 \cdot \mathbb{C}_{\mathsf{PBS}} + \mathbb{C}_{\mathsf{CBS}}$.

Lemma 14 ($\mathbb{C}^{\ell_\mathfrak{m}}_{\mathsf{IntMul}}$) *To simplify the algorithm, the operation* IntMul *proposed in the algorithm, have a complexity which can be bound by* $\left(2 \cdot {\ell_\mathfrak{m}}^2 + {\ell_\mathfrak{m}}^2/\rho_\mathfrak{m}\right) \cdot \mathbb{C}_{\mathsf{PBS}}$. *However, in our implementation, we use a slight modification of this algorithm to remove unnecessary computations (the part of the multiplication which does not appear in the final mantissa). In practice, the complexity is bounded by* $(2 \cdot (\ell_\mathfrak{m}/2 + 1)^2 + \ell_\mathfrak{m} \cdot (\ell_\mathfrak{m}/2 + 1)/\rho_\mathfrak{m}) \cdot \mathbb{C}_{\mathsf{PBS}}$.

Proof 13 (Proof of complexity $\mathbb{C}^{\ell_\mathfrak{m}}_{\mathsf{IntMul}}$) *The mantissa* $\mathfrak{m}$ *represents a value in* $\left[2^{\rho_\mathfrak{m}\cdot(\ell_\mathfrak{m}-1)}, 2^{\rho_\mathfrak{m}\cdot\ell_\mathfrak{m}} - 1\right]$. *So* $\mathfrak{m}^2$ *is in*

$\left[2^{2\cdot\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)}, 2^{2\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}} - 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}+1} + 1\right]$. *After a multiplication, the smallest reachable value is* $2^{2\cdot\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)}$ *and as we keep only the* $\ell_{\mathfrak{m}}$ *most significant blocks, all the values smaller than* $2^{2\cdot\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-1)-\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}} = 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}-2\rho_{\mathfrak{m}}}$ *are lost so we don't need to compute them. These values correspond to the part of the mantissa* $\mathfrak{m}'$ *such that* $\mathfrak{m}'^2 < 2^{\rho_{\mathfrak{m}}\cdot\ell_{\mathfrak{m}}-2\rho_{\mathfrak{m}}}$ *so the part* $\mathfrak{m}'$ *such that* $\mathfrak{m}' < 2^{\rho_{\mathfrak{m}}\cdot(\ell_{\mathfrak{m}}-2)/2}$. *The part of the mantissa* $\mathfrak{m}'$ *corresponds to the* $\ell_{\mathfrak{m}}/2-1$ *less significant blocks of the mantissa* $\mathfrak{m}$. *So we only need to compute the multiplication on the* $\ell_{\mathfrak{m}}/2+1$ *most significant blocks.*

Proof 14 (Correctness of Multiplication (Algorithm 10)) *In the first step of the operation, we compute the sign of the multiplication by summing the two signs* $\mathfrak{s}_1$ *and* $\mathfrak{s}_2$. *As the sign is on the padding bit, the addition is done modulo 2. If the signs are equal, after the operation, the sign is positive (*$\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$*), otherwise the sign is negative (*$\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(1)$*).*

Next, we compute the multiplication of the two mantissas. IntMul *return the product of two integers. If* $\mathfrak{m}_1 \neq 0$ *and* $\mathfrak{m}_2 \neq 0$, *we have* $\mathfrak{m}_i \in \left[2^{(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}} - 1\right]$ *for* $i \in \{0,1\}$, *so* $\mathfrak{m}_1 \cdot \mathfrak{m}_2 \in \left[2^{2\cdot(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{2\cdot\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}} - 2^{\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}+1} + 1\right]$ *(note that the value is stored in* $2\cdot\ell_{\mathfrak{m}}$ *blocks). As we want to keep the classical representation of the mantissa (*$\ell_{\mathfrak{m}}$ *blocks, where the most significant block is non-zero except when the result equals zero), we will remove the least significant blocks, ensuring that only the* $\ell_{\mathfrak{m}}$ *most significant blocks remain.*

To do so, we distinguish two cases after the multiplication. The case where $\mathfrak{m}_1 \cdot \mathfrak{m}_2 \in \left[2^{2\cdot(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{(2\cdot\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}\right)$ *(the most significant block after the multiplication contains zero) and the case where* $\mathfrak{m}_1 \cdot \mathfrak{m}_2 \in \left[2^{(2\cdot\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{2\cdot\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}} - 2^{\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}+1} + 1\right]$. *During the operation* IntMul, *the carry buffer of each ciphertext in* $\mathbf{ct}_{\mathfrak{m}_{\mathsf{mul}}}$ *is emptied. As the carry buffer of* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1}$ *and* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-2}$ *are empty, by multiplying* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1}$ *by* $2^{\rho_{\mathfrak{m}}}$, *we have* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1}\cdot 2^{\rho_{\mathfrak{m}}} \in \{0\}\cup[2^{\rho_{\mathfrak{m}}}, 2^{2\cdot\rho_{\mathfrak{m}}})$. *As* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-2} \in [0, 2^{\rho_{\mathfrak{m}}})$, *we can sum these two values such that* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1} = \mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1}\cdot 2^{\rho_{\mathfrak{m}}} + \mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-2}$ *(Line 3). We have now* $\mathfrak{m}_1\cdot\mathfrak{m}_2 \in \left[2^{2\cdot(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{2\cdot\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}} - 2^{\ell_{\mathfrak{m}}\cdot\rho_{\mathfrak{m}}+1} + 1\right]$ *stored in* $2\cdot\ell_{\mathfrak{m}}-1$ *blocks. Now, we remove the* $\ell_{\mathfrak{m}}-1$ *less significant block of the multiplication which represents too small values for the mantissa precision and the most significant block which have its information already stored in the second most significant block* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{mul}},2\ell_{\mathfrak{m}}-1}$. *We obtain* $\mathfrak{m}_{\mathsf{res}} = \left\lfloor \mathfrak{m}_1\cdot\mathfrak{m}_2/2^{(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}\right\rfloor \in \left[2^{(\ell_{\mathfrak{m}}-1)\cdot\rho_{\mathfrak{m}}}, 2^{(\ell_{\mathfrak{m}}+1)\cdot\rho_{\mathfrak{m}}}\right)$ *(Line 4). (In practice, we have modified the algorithm*

Algorithm 10: $\mathsf{CT}_{\mathfrak{f}_{\text{res}}} \leftarrow \mathsf{Multiplication}(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB})$

Context: $\begin{cases} \mathsf{LUT}_{\mathfrak{m}}: & \text{LUT to return 1 if the value equals 0; 0 otherwise} \\ \mathsf{LUT}_{\mathfrak{e}}: & \text{LUT to return 0 if } \left(x \geq 2^{\rho_{\mathfrak{e}}-1}\right); \text{ 1 otherwise} \end{cases}$

Input:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}_1} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_1} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_1) \\ \mathbf{ct}_{\mathfrak{e}_1} = [\mathsf{ct}_{\mathfrak{e}_1,\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_1,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{1_{\ell_{\mathfrak{e}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{1_0})\right] \\ \mathbf{ct}_{\mathfrak{m}_1} = [\mathsf{ct}_{\mathfrak{m}_1,\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_1,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{1_{\ell_{\mathfrak{m}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{1_0})\right] \end{cases} \\ \mathbf{ct}_{\mathfrak{f}_2} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_2) \\ \mathbf{ct}_{\mathfrak{e}_2} = [\mathsf{ct}_{\mathfrak{e}_2,\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_2,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{2_{\ell_{\mathfrak{e}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{2_0})\right] \\ \mathbf{ct}_{\mathfrak{m}_2} = [\mathsf{ct}_{\mathfrak{m}_2,\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_2,0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{2_{\ell_{\mathfrak{m}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{2_0})\right] \end{cases} \\ \mathsf{PUB}: \text{Public materials for } \textbf{PBS-KS} \text{ and for } \mathsf{CBS} \end{cases}$$

Output:
$$\begin{cases} \mathbf{ct}_{\mathfrak{f}_{\text{res}}} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_{\text{res}}} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_{\text{res}}) \\ \mathbf{ct}_{\mathfrak{e}_{\text{res}}} = [\mathsf{ct}_{\mathfrak{e}_{\text{res}},\ell_{\mathfrak{e}}-1}, ..., \mathsf{ct}_{\mathfrak{e}_{\text{res}},0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\text{res}_{\ell_{\mathfrak{e}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\text{res}_0})\right] \\ \mathbf{ct}_{\mathfrak{m}_{\text{res}}} = [\mathsf{ct}_{\mathfrak{m}_{\text{res}},\ell_{\mathfrak{m}}-1}, ..., \mathsf{ct}_{\mathfrak{m}_{\text{res}},0}] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_{\text{res}_{\ell_{\mathfrak{m}}-1}}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{\text{res}_0})\right] \end{cases} \end{cases}$$

1 $\mathsf{ct}_{\mathfrak{s}_{\text{res}}} \leftarrow \mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2}$
2 $\mathbf{ct}_{\mathfrak{m}_{\text{mul}}} = [\mathsf{ct}_{\mathfrak{m}_{\text{mul}},2\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_{\text{mul}},0}] \leftarrow \mathsf{IntMul}(\mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}_{\mathfrak{m}_2}, \mathsf{PUB})$
3 $\mathsf{ct}_{\mathfrak{m}_{\text{mul}},2\ell_{\mathfrak{m}}-2} \leftarrow \mathsf{ct}_{\mathfrak{m}_{\text{mul}},2\ell_{\mathfrak{m}}-1} \cdot 2^{\rho_{\mathfrak{m}}} + \mathsf{ct}_{\mathfrak{m}_{\text{mul}},2\ell_{\mathfrak{m}}-2}$
4 $\mathbf{ct}_{\mathfrak{m}_{\text{res}}} = [\mathsf{ct}_{\mathfrak{m}_{\text{res}},\ell_{\mathfrak{m}}-1}, ...\mathsf{ct}_{\mathfrak{m}_{\text{res}},0}] \leftarrow [\mathsf{ct}_{\mathfrak{m}_{\text{mul}},2\ell_{\mathfrak{m}}-2}, ...\mathsf{ct}_{\mathfrak{m}_{\text{mul}},\ell_{\mathfrak{m}}-1}]$
/* $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{m}}} \in \mathsf{LWE}_{\boldsymbol{s}}(1)$ if $\mathsf{ct}_{\mathfrak{m}_{\text{res}},\ell_{\mathfrak{m}}-1} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$; $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$ otherwise */
5 $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{m}}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\mathfrak{m}_{\text{res}},\ell_{\mathfrak{m}}-1}, \mathsf{LUT}_{\mathfrak{m}}, \mathsf{PUB})$
6 $\mathbf{ct}_{\mathfrak{e}_{\text{res}}} = [\mathsf{ct}_{\mathfrak{e}_{\text{res}},\ell_{\mathfrak{e}}-1}, ...\mathsf{ct}_{\mathfrak{e}_{\text{res}},0}] \leftarrow \mathsf{IntAdd}(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathsf{PUB})$
/* $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$ if $\mathsf{ct}_{e_{\text{res}},\ell_{\mathfrak{e}}-1} \in \mathsf{LWE}_{\boldsymbol{s}}(x)$ with $x \geq 2^{\rho_{\mathfrak{e}}-1}$;
$\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}} \in \mathsf{LWE}_{\boldsymbol{s}}(1)$ otherwise */
7 $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{e_{\text{res}},\ell_{\mathfrak{e}}-1}, \mathsf{LUT}_{\mathfrak{e}}, \mathsf{PUB})$
/* bias have been choose such that $\text{bias} - \ell_{\mathfrak{m}} + 1$ is equal to $2^{\ell_{\mathfrak{e}} \cdot \rho_{\mathfrak{e}}-1}$ so this subtraction can be done only on the most significant block */
8 $\mathsf{ct}_{\mathfrak{e}_{\text{res}},\ell_{\mathfrak{e}}-1} \leftarrow \mathsf{ct}_{\mathfrak{e}_{\text{res}},\ell_{\mathfrak{e}}-1} - \mathsf{TrivialEncrypt}(2^{\rho_{\mathfrak{e}}-1}, 1)$
/* $\mathsf{ct}_{\mathsf{tmp}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$ if $\mathfrak{e}$ is big enough and the most significant mantissa LWE is not null */
9 $\mathsf{ct}_{\mathsf{tmp}} \leftarrow \mathsf{ct}_{\mathsf{tmp}_{\mathfrak{m}}} + \mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}}$
/* encrypt 0 if $\mathsf{ct}_{\mathsf{tmp}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$, 1 otherwise */
10 $\overline{\overline{\mathsf{CT}}} \leftarrow \mathsf{CBS}(\mathsf{ct}_{\mathsf{tmp}}, \mathsf{PUB})$
11 $\mathbf{ct}_{\mathfrak{e}_{\text{res}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}_{\text{res}}}, \mathsf{TrivialEncrypt}(0, \ell_{\mathfrak{e}}), \overline{\overline{\mathsf{CT}}}\right)$,
$\mathbf{ct}_{\mathfrak{m}_{\text{res}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_{\text{res}}}, \mathsf{TrivialEncrypt}(0, \ell_{\mathfrak{m}}), \overline{\overline{\mathsf{CT}}}\right)$; /* Algo 5 */
12 **return** $\mathbf{ct}_{\text{res}} = (\mathsf{ct}_{\mathfrak{s}_{\text{res}}}; \mathbf{ct}_{\mathfrak{e}_{\text{res}}}; \mathsf{CT}_{\mathfrak{m}_{\text{res}}})$

IntMul *such that the carry propagation of the most significant block is not done and such that the useless parts of the multiplication are not computed*

(Proof 13)). In the special case where $\mathfrak{m}_1 = 0$ *or* $\mathfrak{m}_2 = 0$*, the previous step has no impact and* IntMul *will return the value* 0 *on each block. To distinguish the two cases, line 5, a* PBS *checks if the most significant block of the mantissa is equal to zero, such that* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{m}}$ *is in* $\mathsf{LWE}_{\boldsymbol{s}}(1)$ *if* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{res},\ell_\mathfrak{m}-1}}$ *is in* $\mathsf{LWE}_{\boldsymbol{s}}(0)$*, otherwise it returns* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{m}}$ *in* $\mathsf{LWE}_{\boldsymbol{s}}(0)$*.*

Now, we need to update the exponent $\mathfrak{e}$*. First we will sum the two exponents (Line 6). Next, as the exponent has the shape* $\mathfrak{e}'_i + \mathsf{bias}$ *after the sum we obtain* $\mathfrak{e}_1 + \mathfrak{e}_2 = \mathfrak{e}_{\mathsf{res}} = \mathfrak{e}'_{\mathsf{res}} + 2 \cdot \mathsf{bias}$*. So to keep the same representation, we need to remove one* bias*. Moreover, in the previous step, we have removed the* $\ell_\mathfrak{m} - 1$ *blocks of the mantissa, so we need to add* $\ell_\mathfrak{m} - 1$ *to the exponent. In Section 2.5, we chose the bias such that* $\mathsf{bias} - \ell_\mathfrak{m} + 1 = 2^{\ell_\mathfrak{e} \cdot \rho_\mathfrak{e} - 1}$ *so we only need to subtract* $2^{\ell_\mathfrak{e} \cdot \rho_\mathfrak{e} - 1}$ *from the sum of the exponents. Line 4, a* PBS *checks if the exponent is big enough and return an* LWE *ciphertext such that* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{e}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$ *if* $\mathsf{ct}_{\mathfrak{e}_{\mathsf{res},\ell_\mathfrak{e}-1}}$ *is in* $\mathsf{LWE}_{\boldsymbol{s}}(x)$ *with* $x \geq 2^{\rho_\mathfrak{e}-1}$*, otherwise it returns* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{e}} \in \mathsf{LWE}_{\boldsymbol{s}}(1)$*. We can now subtract* $2^{\ell_\mathfrak{e} \cdot \rho_\mathfrak{e} - 1}$ *from the sum of the exponent (Line 8). This operation impacts only the most significant block of the exponent and can be performed directly.*

Now, looking at the two previous control LWE *ciphertexts (*$\mathsf{ct}_{\mathsf{tmp}_\mathfrak{m}}$ *and* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{e}}$*), by summing these two values, we obtain* $\mathsf{ct}_{\mathsf{tmp}_\mathfrak{m}} + \mathsf{ct}_{\mathsf{tmp}_\mathfrak{e}} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$ *if* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{res},\ell_\mathfrak{m}-1}} \notin \mathsf{LWE}_{\boldsymbol{s}}(0)$ *and* $\mathsf{ct}_{\mathfrak{e}_{\mathsf{res},\ell_\mathfrak{e}-1}} \in \mathsf{LWE}_{\boldsymbol{s}}(x)$ *with* $x \geq 2^{\rho_\mathfrak{e}-1}$*. Otherwise, one of the two conditions to perform the multiplication is unmet and the multiplication is not feasible. By using a circuit bootstrapping, we obtain a* GGSW *ciphertext* $\overline{\overline{\mathsf{CT}}}$ *such that* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(0)$ *if we can perform the multiplication and* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(1)$ *otherwise. With the last 2 lines, if* $\overline{\overline{\mathsf{CT}}}$ *is in* $\mathsf{GGSW}^{\beta,\ell}_{\boldsymbol{S}}(0)$*, the algorithm returns the result of the multiplication, otherwise the multiplication is not doable and it returns zero.*

5.2 Division

This operation computes the division of two HFPs. Following this procedure, it is necessary to apply the CarryPropagateFloat algorithm (Algorithm 6). This step will ensure that the homomorphic floating-point number is returned to its standard representation, aligning it with the conventional HFP formalism.

Lemma 15 (Division (Algorithm 11)) *Let* $\mathbf{ct}_{\mathfrak{f}_i}$ *such that* $\mathsf{ct}_{\mathfrak{s}_i} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_i)$*,* $\mathbf{ct}_{\mathfrak{e}_i} = \left[\mathsf{ct}_{\mathfrak{e}_{i,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{i,0}}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{i,\ell_\mathfrak{e}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{i,0}\right)\right]$

encrypting $\mathfrak{e} < 2^{\rho_\mathfrak{e}}$ *and* $\mathbf{ct}_{\mathfrak{m}_i} = \left[\mathsf{ct}_{\mathfrak{m}_{i,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{i,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,\ell_\mathfrak{m}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{i,0})]$ *encrypting* $\mathfrak{m} < 2^{\rho_\mathfrak{m}}$ *with* $i \in \{1,2\}$ *be two ciphertexts encrypting* $\mathfrak{f}_i = (-1)^{\mathfrak{s}_i} \cdot \mathfrak{m}_i \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_i - \mathsf{bias}}$. *Let* $(\mathbf{ct}_{\mathfrak{m}_\mathsf{res}}, \mathbf{ct}_{\mathfrak{e}_\mathsf{res}}, \mathsf{ct}_{\mathfrak{s}_\mathsf{res}}) = \mathbf{ct}_{\mathfrak{f}_\mathsf{res}} \leftarrow \mathsf{Division}(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB})$. *Then,* $\mathsf{DecryptFloat}(\mathbf{ct}_{\mathfrak{f}_\mathsf{res}}) = \mathfrak{f}_\mathsf{res} = (-1)^{\mathfrak{s}_\mathsf{res}} \cdot \mathfrak{m}_\mathsf{res} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_\mathsf{res} - \mathsf{bias}} = \mathfrak{f}_1/\mathfrak{f}_2 + \epsilon$ *with* $|\epsilon| < (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_\mathsf{res} - \mathsf{bias}}$ *s.t.* $\mathsf{ct}_{\mathfrak{s}_\mathsf{res}} = \mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2}$. *If* $\mathsf{ct}_{\mathfrak{e}_2} < \mathsf{ct}_{\mathfrak{e}_1} + \mathsf{bias} + \ell_\mathfrak{m} - 1$, *then* $\mathsf{ct}_{\mathfrak{m}_\mathsf{res}} = \left\lfloor \mathsf{ct}_{\mathfrak{m}_1} \cdot 2^{\ell_\mathfrak{m}-1} / \mathsf{ct}_{\mathfrak{m}_2} \right\rfloor$ *and* $\mathsf{ct}_{\mathfrak{e}_\mathsf{res}} = \mathsf{ct}_{\mathfrak{e}_1} + \mathsf{bias} + \ell_\mathfrak{m} - 1 - \mathsf{ct}_{\mathfrak{e}_2}$. *Else,* $\mathsf{ct}_{\mathfrak{e}_\mathsf{res}} = 0$ *and* $\mathsf{ct}_{\mathfrak{m}_\mathsf{res}} = 0$. *The complexity of the algorithm is:* $\mathbb{C}_\mathsf{Division} = \mathbb{C}^{\ell_\mathfrak{e}}_{\mathsf{IntSub}^*} + \mathbb{C}^{2\cdot\ell_\mathfrak{m}}_\mathsf{IntDiv} + \mathbb{C}_\mathsf{CBS}$.

Algorithm 11: $\mathbf{ct}_\mathfrak{f} \leftarrow \mathsf{Division}(\mathbf{ct}_{\mathfrak{f}_1}, \mathbf{ct}_{\mathfrak{f}_2}, \mathsf{PUB})$

Input:

$\mathbf{ct}_{\mathfrak{f}_1} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_1} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_1) \\ \mathbf{ct}_{\mathfrak{e}_1} = \left[\mathsf{ct}_{\mathfrak{e}_{1,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{1,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{1,\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{1,0})] \\ \mathbf{ct}_{\mathfrak{m}_1} = \left[\mathsf{ct}_{\mathfrak{m}_{1,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{1,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{1,\ell_\mathfrak{m}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{1,0})] \end{cases}$

$\mathbf{ct}_{\mathfrak{f}_2} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_2) \\ \mathbf{ct}_{\mathfrak{e}_2} = \left[\mathsf{ct}_{\mathfrak{e}_{2,\ell_\mathfrak{e}-1}}, \cdots, \mathsf{ct}_{\mathfrak{e}_{2,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{2,\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{2,0})] \\ \mathbf{ct}_{\mathfrak{m}_2} = \left[\mathsf{ct}_{\mathfrak{m}_{2,\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_{2,0}}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{2,\ell_\mathfrak{m}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{2,0})] \end{cases}$

PUB : Public materials for **PBS-KS** and for CBS

Output:

$\mathbf{ct}_{\mathfrak{f}_\mathsf{res}} = \begin{cases} \mathsf{ct}_{\mathfrak{s}_\mathsf{res}} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_\mathsf{res}) \\ \mathbf{ct}_{\mathfrak{e}_\mathsf{res}} = [\mathsf{ct}_{\mathfrak{e}_\mathsf{res},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_\mathsf{res},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\mathsf{res}_{\ell_\mathfrak{e}-1}}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\mathsf{res}_0})] \\ \mathbf{ct}_{\mathfrak{m}_\mathsf{res}} = [\mathsf{ct}_{\mathfrak{m}_\mathsf{res},\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_\mathsf{res},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{\mathsf{res}_{\ell_\mathfrak{m}-1}}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{\mathsf{res}_0})] \end{cases}$

$\mathsf{ct}_{\mathfrak{s}_\mathsf{res}} \leftarrow \mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2}$

$\mathsf{ct}_{\mathfrak{e}_1} \leftarrow \mathsf{ct}_{\mathfrak{e}_1} + \mathsf{TrivialEncrypt}(\mathsf{bias} + \ell_\mathfrak{m} - 1, \ell_\mathfrak{e})$

$(\mathsf{ct}_{\mathfrak{e}_\mathfrak{s}}, \mathbf{ct}_\mathfrak{e} = [\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}}, \cdots, \mathsf{ct}_{\mathfrak{e},0}]) \leftarrow \mathsf{IntSub}^*(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{e}_2}, \mathsf{PUB})$

$\mathsf{ct}_{\mathfrak{m}_1} = [\mathsf{ct}_{\mathfrak{m}_1,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_1,0}, \mathsf{ct}_0, \cdots, \mathsf{ct}_0] \leftarrow \mathsf{ct}_{\mathfrak{m}_1} || \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{m} - 1)$

$\mathsf{ct}_{\mathfrak{m}_2} = [\mathsf{ct}_0, \cdots, \mathsf{ct}_0, \mathsf{ct}_{\mathfrak{m}_2,\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_2,0}] \leftarrow \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{m} - 1) || \mathsf{ct}_{\mathfrak{m}_2}$

$\mathsf{ct}_{\mathfrak{m}_\mathsf{div}} = [\mathsf{ct}_{\mathfrak{m}_\mathsf{div},2\cdot\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m}_\mathsf{div},0}] \leftarrow \mathsf{IntDiv}(\mathsf{ct}_{\mathfrak{m}_1}, \mathsf{ct}_{\mathfrak{m}_2}, \mathsf{PUB})$

$\mathsf{ct}_{\mathfrak{m}_\mathsf{div},2\ell_\mathfrak{m}-2} \leftarrow \mathsf{ct}_{\mathfrak{m}_\mathsf{div},2\ell_\mathfrak{m}-1} \cdot 2^{\rho_\mathfrak{m}} + \mathsf{ct}_{\mathfrak{m}_\mathsf{div},2\ell_\mathfrak{m}-2}$

$\mathbf{ct}_{\mathfrak{m}_\mathsf{res}} = [\mathsf{ct}_{\mathfrak{m}_\mathsf{res},\ell_\mathfrak{m}-1}, ...\mathsf{ct}_{\mathfrak{m}_\mathsf{res},0}] \leftarrow [\mathsf{ct}_{\mathfrak{m}_\mathsf{div},2\ell_\mathfrak{m}-2}, ...\mathsf{ct}_{\mathfrak{m}_\mathsf{div},\ell_\mathfrak{m}-1}]$

$\overline{\overline{\mathsf{CT}}}_{\mathfrak{e}_\mathfrak{s}} \leftarrow \mathsf{CBS}(\mathsf{ct}_{\mathfrak{e}_\mathfrak{s}}, \mathsf{PUB})$

$\mathbf{ct}_{\mathfrak{e}_\mathsf{res}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_\mathfrak{e}, \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{e}), \overline{\overline{\mathsf{CT}}}_{\mathfrak{e}_\mathfrak{s}}\right)$,

$\mathbf{ct}_{\mathfrak{m}_\mathsf{res}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_\mathsf{res}}, \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{m}), \overline{\overline{\mathsf{CT}}}_{\mathfrak{e}_\mathfrak{s}}\right)$; `/* Algo 5 */`

return $\mathbf{ct}_\mathfrak{f} = (\mathsf{ct}_{\mathfrak{s}_\mathsf{res}}, \mathbf{ct}_{\mathfrak{e}_\mathsf{res}}, \mathbf{ct}_{\mathfrak{m}_\mathsf{res}})$

Proof 15 (Correctness of Division (Algorithm 11)) *On the first line, we compute the output sign. If the signs are equal, the sign after the operation is positive ($\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$), otherwise the sign is negative ($\mathsf{ct}_{\mathfrak{s}_1} + \mathsf{ct}_{\mathfrak{s}_2} \in \mathsf{LWE}_{\boldsymbol{s}}(1)$).*

The algorithm IntDiv *takes two vectors of ciphertexts that represent two integers and returns the quotient of the division. In our context, we can not divide directly the mantissas. In fact, the mantissas are in the same interval and very close, if we were to divide them directly, the quotient would only be a value of a few digits. In our case, we want a value in the interval $\left[2^{\rho_\mathfrak{m}\cdot(\ell_\mathfrak{m}-1)}, 2^{\rho_\mathfrak{m}\cdot\ell_\mathfrak{m}}\right)$. To get a result in the right interval, we add some blocks encrypting zeros after the blocks of the first mantissa $\mathfrak{m}_1$ (Line 4). Adding the zeros to $\mathbf{ct}_{\mathfrak{m}_1}$ corresponds to compute $\mathfrak{m}_1 \cdot 2^{\ell_\mathfrak{m}\cdot\rho_\mathfrak{m}} \in \left[2^{\rho_\mathfrak{m}\cdot(2\ell_\mathfrak{m}-1)}, 2^{\rho_\mathfrak{m}\cdot 2\ell_\mathfrak{m}}\right)$. Now if we divide $\mathfrak{m}_1 \cdot 2^{\ell_\mathfrak{m}\cdot\rho_\mathfrak{m}}$ by $\mathfrak{m}_2$, we will obtain a result in $\left[2^{(\ell_\mathfrak{m}-1)\cdot\rho_\mathfrak{m}}, 2^{(\ell_\mathfrak{m}+1)\cdot\rho_\mathfrak{m}}\right)$. As explained in Proof 14, this value can be stored in $\ell_\mathfrak{m}$ blocks if we use the carry buffer of the most significant block. The carry buffer will be later cleaned during the call to CarryPropagateFloat (Algorithm 6). So after the shift of the mantissa (Line 4) and the division (Line 6), we obtain a new mantissa in the interval $\left[2^{(\ell_\mathfrak{m}-1)\cdot\rho_\mathfrak{m}}, 2^{(\ell_\mathfrak{m}+1)\cdot\rho_\mathfrak{m}}\right)$.*

After the division of the mantissa, we need to update the exponent. To do so, we need to subtract (Line 3) the two exponents, then add the bias and finally add the number of trivial ciphertexts added in Line 4. If the subtraction of the exponent returns a negative result (Line 3, $\mathsf{ct}_{\mathfrak{e}_s} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$), the division can not be done. In this case, Algorithm 11 returns the value zero (Line 10 and 11), otherwise it returns the result of the division of the two floating-point numbers.

6 Experimental Results

In this section, we demonstrate the practicability of our results by providing all cryptographic parameters, encodings, and both sequential and parallel timings.

Encodings In Table 3, we describe the different encodings used to represent 64, 32, 16 and 8 bits floating-point numbers (Sec. 3.2) in the homomorphic world. Research in [BBB+22] indicates that a 4-bit precision message leads to the best precision-cost ratio; therefore, we focus on representations with $\rho_\mathfrak{m} = \rho_\mathfrak{e} = 2$. However, variations with $\rho_\mathfrak{m} \neq \rho_\mathfrak{e}$ may yield better tim-

ings depending on the specific use case. Additionally, in Table 3, we give the encoding for the TFHE-minifloats encoded over 8 bits as detailed in Sec. 3.1. For the TFHE-Minifloats, the value of the bias does not impact the timings and can be freely chosen.

	$\ell_{\mathfrak{m}}$	$\rho_{\mathfrak{m}}$	$\ell_{\mathfrak{e}}$	$\rho_{\mathfrak{e}}$	bias
TFHE_FP$_{64b}$	27	2	5	2	539
TFHE_FP$_{32b}$	13	2	4	2	140
TFHE_FP$_{16b}$	6	2	3	2	37
TFHE_FP$_{8b}$	3	2	2	2	10
TFHE-Minifloat$^{\rho=4}$	3	$\varnothing$	4	$\varnothing$	8
TFHE-Minifloat$^{\rho=2}$	3	$\varnothing$	4	$\varnothing$	8

Table 3: Encodings for HFP and Minifloats

Parameter Selection In Lemma 12, we found two noise constraints that the parameters must satisfy in order to guarantee the correctness of Algorithm 9. We applied the same reasoning to Algorithm 10 and found that all the additional noise constraints are dominated by the constraints introduced in Lemma 12. It means that parameters that satisfy the constraints of Algorithm 9 will also satisfy the constraints of Algorithm 10.

As explained in Lemmas 11 and 13, the number of PBSs in each algorithm is different and this has an impact on the failure probability of each algorithm. We followed [BBB+22]'s blueprint to compute the individual PBS failure probability using the number of dominant PBS in each algorithm. Using the parameters presented in Table **??**, the maximal failure probability for the homomorphic addition and for the homomorphic multiplication are respectively $2^{-13.9}$ and $2^{-12.8}$ (note that the failure probability of one KS-PBS is smaller than 2^{-40}). We tested these parameters on a chain of a hundred operations on random inputs with random operations without detecting any errors due to the noise, only errors due to floating-point approximations.

Timings All of our experiments have been carried out on AWS with a m6i.metal instance Intel Xeon 8375C (Ice Lake) at 3.5 GHz, with 128 vCPUs and 512.0 GiB of memory using the TFHE-rs library [Zam22]. Our code is available here[1]. In Table 5, we give the timings in seconds for all the

[1] https://github.com/zama-ai/tfhe-rs/tree/artifact_tches_2025

arithmetic operations (i.e., add, sub, mul, div) and the ReLU and Sigmoid functions. Both sequential and parallel timings are given when possible (e.g. the division over integers is only implemented in parallel in TFHE-rs). Note that all arithmetic operations are followed by a carry propagation, which is obviously taken into account in the timings.

Finally, in Table 6, we present the timings for the WoP-PBS based approach. Although the multiplication timings are quite similar between HFP and minifloats, the addition operations perform significantly better using the WoP-PBS. This means that when computing with 8-bit floats, using the minifloats is generally better. Any larger precision requires to run the HFP method, whose timings show that many circuits based on floats can be practically evaluated for the first time. Only the division operation cannot be considered as practical. Note that the division is suffering from the slowness of the division over the integers, and not really from the approach described in here.

		Addition (Alg.9 & 6)	Multiplication (Alg.10 & 6)	Division (Alg.11 & 6)	Sigmoid (Alg.13)	Relu (Alg.12)
TFHE_FP$_{64b}$	Sequential	12.32 s	87.15 s	∅	0.342 s	0.122 s
	Parallel	3.98 s	2.26 s	39.75 s	∅	∅
TFHE_FP$_{32b}$	Sequential	7.10 s	20.57 s	∅	0.342 s	0.120 s
	Parallel	2.50 s	1.03 s	15.18 s	∅	∅
TFHE_FP$_{16b}$	Sequential	3.89 s	3.83 s	∅	0.361 s	0.155 s
	Parallel	1.52 s	0.558 s	4.34 s	∅	∅
TFHE_FP$_{8b}$	Sequential	2.21 s	1.19 s	∅	0.388 s	0.153 s
	Parallel	1.13 s	0.444 s	1.76 s	∅	∅

Table 5: Timings of the HFP depending on the precision.

	TFHE-Minifloat$^{\rho=4}$	TFHE-Minifloat$^{\rho=2}$
Bivariate Operation (e.g., add, mul, ...)	1.2819 s	0.9957 s

Table 6: Timings for the 8 bit representations of the TFHE-Minifloat.

In the previous table, we present benchmarks obtained with a failure probability around 2^{-14}. To better evaluate our new algorithms, we also include benchmarks of the addition and the multiplication with a failure probability of around 2^{-40} (see Table 7). We observe that reducing the

failure probability has only a minor impact on execution time which proves that our contribution scales well with small failure probabilities.

		Add	**Mul**
$\mathsf{TFHE\_FP}_{\mathsf{32b}}$	Sequential	7.49s	23.2s
$\mathsf{TFHE\_FP}_{\mathsf{32b}}$	Parallelized	2.83s	1.24s
$\mathsf{TFHE\_FP}_{\mathsf{64b}}$	Sequential	13.3s	98.2s
$\mathsf{TFHE\_FP}_{\mathsf{64b}}$	Parallelized	4.28s	2.75s

Table 7: Performance for Addition and Multiplication using $\mathsf{pfail} \leq 2^{-40}$

7 More Features over HFP

This section extends our approach to cover a wider range of practical applications by adding the support of special values and efficient approximate functions.

7.1 Managing Special Values

In the classical floating-point arithmetic, when a value overflows the highest bound of the exponent, this value can not be represented anymore. In this case, the floating-point reaches the infinity. As previously described, our algorithms do not manage the values plus/minus infinity and Not a Number (NaN), but as we show now, they can easily be extended to do so. The idea is to add two encrypted Booleans to represent $+\infty$ and $-\infty$ such that: if only one is set, then it means that we have reached plus (resp. minus) infinity; if both are set, the value is interpreted as NaN. During an operation, the infinity value is reached as soon as an overflow occurs on the exponent, i.e., if the carry of the most significant block of the exponent is not empty. This check is done by computing a simple PBS on $\mathsf{ct}_{\mathsf{res},\ell_\mathfrak{e}-1}$, which returns a flag encrypting 0 if the carry is empty, or 1 otherwise. This flag is then given as an additional input to the $\mathsf{CarryPropagateFloat}$ (Alg. 6). Then, by computing a CBS on the sign, this will return the correct sign of the flag, i.e., plus or minus infinity (or 0). This process ends with the computation of a simple CMux tree which will properly update Booleans ciphertext depending on the flag value. Regarding the support of special values, the overhead is linear in the number of blocks composing the HFP. Thus, in comparison with the

numbers of PBS or CBS needed to perform operations without any special values, the overhead should be negligible.

7.2 Computing Function Approximations

Beyond the arithmetic operations, floating-point numbers are particularly convenient to compute approximations of complex functions, via the Taylor series. A Taylor series of a real function $f(x)$ that is infinitely differentiable at a real number a is the power series $\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$, where $n!$ denotes the factorial of n and $f^{(n)}(a)$ denotes the n-th derivative of f evaluated at the point a. When $a = 0$, this is called a Maclaurin serie and takes the form $\sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!}x^n$. The Maclaurin method is advantageous when working with homomorphic floating-point numbers, given that the value of $f(a)$ is not known. Computing such a serie is a direct application of our method, as each of its term can be computed using the previously defined arithmetic operators. Another advantage is that the needed precision and the computational time can be adjusted to fit the use case, i.e., by changing the value of n. For instance, we have practically computed $\sin(x) \approx \sum_{n=0}^{2} \frac{(-1)^n}{(2n+1)!}x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!}$ and $\cos(x) \approx \sum_{n=0}^{2} \frac{(-1)^n}{(2n)!}x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$, which gives good results for values of $x \in [-1, 1]$. In Table 8, we present numerical values obtained from our approximations of the cosine and sine functions using the Maclaurin series.

	Cos(0.9636989235877991)	Sin(0.41880202293395996)
Exact value (64 bits)	0.5704859425112639	0.4066663011129846
Approximate value (64 bits)	**0.5715802311897278**	**0.4066667483866177**
Approximate value (32 bits)	**0.57158023**	**0.4066667**6
This work (32bits)	**0.5715802311897278**	**0.4066667**5567626953

Table 8: Result obtain for the cosinus and sinus with Maclaurin series. The bold digits are the one which are equal to the digits of the approximate result of the clear double precision value.

7.3 Other Operations

With the homomorphic floating-point representation, we can efficiently support usual functions used in machine learning. To evaluate the ReLU function, we can apply a circuit bootstrapping on the sign $\mathfrak{s}$ and return either the

input floating-point or an encryption of zero using a cmux. The complete algorithm is detailed in Algorithm 12 in Section 7.3.

In the same manner, we can evaluate an approximate sigmoid function that returns the identity for values in the interval $[-1, 1]$, and returns the constant value 1 or -1 otherwise. More details are provided in Algorithm 13 in Section 7.3. To be closer to the classical sigmoid, we can combine this approximate sigmoid with the Maclaurin series introduced in Section 7.2. Other classical operations like the minimum, the maximum or the equality between two values can be easily performed on homomorphic floating-point numbers.

7.3.1 ReLU

Lemma 16 (ReLU (Algorithm 12)) *Let* $\mathbf{ct}_\mathfrak{f}$ *such that* $\mathsf{ct}_\mathfrak{s} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s})$,$\mathbf{ct}_\mathfrak{e} = \left[\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_0}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_0)]$ *encrypting* $\mathfrak{e} < 2^{\rho_\mathfrak{e}}$ *and* $\mathbf{ct}_\mathfrak{m} = \left[\mathsf{ct}_{\mathfrak{m}_{\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m}_0}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{\ell_\mathfrak{m}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_0)]$ *encrypting* $\mathfrak{m} < 2^{\rho_\mathfrak{m}}$ *be ciphertexts encrypting* $\mathfrak{f} = (-1)^\mathfrak{s} \cdot \mathfrak{m} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}-\mathsf{bias}}$.

Let $(\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{res}}}) = \mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}} \leftarrow \mathsf{Relu}(\mathbf{ct}_\mathfrak{f}\mathsf{PUB})$. *Then* $\mathsf{DecryptFloat}(\mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}}) = \mathfrak{f}_{\mathsf{res}}$ *with* $\mathfrak{f}_{\mathsf{res}} = (-1)^{\mathfrak{s}_{\mathsf{res}}} \cdot \mathfrak{m}_{\mathsf{res}} \cdot (2^{\rho_\mathfrak{m}})^{\mathfrak{e}_{\mathsf{res}}-\mathsf{bias}}$ *such that if* $\mathsf{ct}_\mathfrak{s} \in \mathsf{LWE}_{\boldsymbol{s}}(0)$, $\mathsf{ct}_{\mathfrak{s}_{\mathsf{res}}} = \mathsf{ct}_\mathfrak{s}$, $\mathsf{ct}_{\mathfrak{e}_{\mathsf{res}}} = \mathsf{ct}_\mathfrak{e}$ *and* $\mathsf{ct}_{\mathfrak{m}_{\mathsf{res}}} = \mathsf{ct}_\mathfrak{m}$ *Else* $\mathbf{ct}_{\mathfrak{f}_{\mathsf{res}}}$ *encrypt zero.*
The complexity of the algorithm is: $\mathbb{C}_{\mathsf{Division}} = \mathbb{C}^{\ell_\mathfrak{e}}_{\mathsf{IntSub}^*} + \mathbb{C}^{2\cdot\ell_\mathfrak{m}}_{\mathsf{IntDiv}} + \mathbb{C}_{\mathsf{CBS}}$

Algorithm 12: $\mathbf{ct}_\mathfrak{f} \leftarrow \mathsf{ReLU}(\mathbf{ct}_\mathfrak{f}, \mathsf{PUB})$

Input:
- $\mathbf{ct}_\mathfrak{f} = \begin{cases} \mathsf{ct}_\mathfrak{s} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}) \\ \mathbf{ct}_\mathfrak{e} = [\mathsf{ct}_{\mathfrak{e},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_0)] \\ \mathbf{ct}_\mathfrak{m} = [\mathsf{ct}_{\mathfrak{m},\ell_\mathfrak{m}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}\ell_\mathfrak{m}-1), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_0)] \end{cases}$
- PUB : Public materials for **PBS-KS** and for CBS

Output:
- $\mathbf{ct}_\mathfrak{f} = \begin{cases} \mathsf{ct}_\mathfrak{s} \in \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{s}_{\mathsf{res}}) \\ \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}} = [\mathsf{ct}_{\mathfrak{e}_{\mathsf{res}},\ell_\mathfrak{e}-1}, \cdots, \mathsf{ct}_{\mathfrak{e}_{\mathsf{res}},0}] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\mathsf{res},\ell_\mathfrak{e}-1}), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{e}_{\mathsf{res},0})] \\ \mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}} = \left[\mathsf{ct}_{\mathfrak{m}_{\mathsf{res},\ell_\mathfrak{m}-1}}, \cdots, \mathsf{ct}_{\mathfrak{m},\mathsf{res},0}\right] \in [\mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}\mathsf{res}, \ell_\mathfrak{m}-1), ..., \mathsf{LWE}_{\boldsymbol{s}}(\mathfrak{m}_{\mathsf{res},0})] \end{cases}$

`/* encrypt 0 if` $sign == 0$`, 1 otherwise */`

1 $\overline{\overline{\mathsf{CT}}} \leftarrow \mathsf{CBS}(\mathsf{ct}_\mathfrak{s}, \mathsf{PUB})$
2 $\mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}} \leftarrow \mathsf{ExtendedCMux}(\mathbf{ct}_\mathfrak{e}, \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{e}), \overline{\overline{\mathsf{CT}}})$
3 $\mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}} \leftarrow \mathsf{ExtendedCMux}(\mathbf{ct}_\mathfrak{m}, \mathsf{TrivialEncrypt}(0, \ell_\mathfrak{m}), \overline{\overline{\mathsf{CT}}})$
4 **return** $\mathbf{ct}_\mathfrak{f} = [\mathsf{ct}_\mathfrak{s}, \mathbf{ct}_{\mathfrak{m}_{\mathsf{res}}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{res}}}]$

Proof 16 (ReLU (Algorithm 12)) *First we use a* CBS *on the sign to obtain a* GGSW *ciphertext such that* $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(0)$ *if* $\mathsf{ct}_{\mathfrak{s}}$ *is in* $\mathsf{LWE}_{\boldsymbol{s}}(0)$*, otherwise,* $\overline{\overline{\mathsf{CT}}}$ *is in* $\mathsf{GGSW}_{\boldsymbol{S}}^{\beta,\ell}(1)$

Next with the GGSW *ciphertext, we return the input ciphertext if the sign is positive otherwise we return zero.*

7.3.2 Approximate Sigmoid

An efficient algorithm to compute an approximation of the sigmoid function compatible with the HFP representation is presented in Algorithm 13.

Algorithm 13: $\mathbf{ct}_{\mathfrak{f}} \leftarrow \mathsf{ApproxSigmoid}\left(\mathbf{ct}_{\mathfrak{f}}, \mathsf{PUB}\right)$

Context: $\begin{cases} \mathsf{LUT}_{\mathfrak{m}}: & \text{LUT to return 1 if the value equals 1, 1 otherwise} \\ \mathsf{LUT}_{\mathfrak{e}}: & \text{LUT to return 0 if } \left(x >= 2^{\rho_{\mathfrak{e}}-1}\right); \text{ 1 otherwise} \end{cases}$

Input: $\begin{cases} \mathbf{ct}_{\mathfrak{f}} = \begin{cases} \mathsf{ct}_{\mathfrak{s}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}\right) \\ \mathbf{ct}_{\mathfrak{e}} = \left[\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)\right] \\ \mathbf{ct}_{\mathfrak{m}} = \left[\mathsf{ct}_{\mathfrak{m},\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}\ell_{\mathfrak{m}}-1\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_0\right)\right] \end{cases} \\ \mathsf{PUB}: \text{Public materials for } \textbf{PBS-KS} \text{ and for} \mathsf{CBS} \end{cases}$

Output: $\begin{cases} \mathbf{ct}_{\mathfrak{f}} = \begin{cases} \mathsf{ct}_{\mathfrak{s}} \in \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{s}\right) \\ \mathbf{ct}_{\mathfrak{e}} = \left[\mathsf{ct}_{\mathfrak{e},\ell_{\mathfrak{e}}-1}, \cdots, \mathsf{ct}_{\mathfrak{e},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_{\ell_{\mathfrak{e}}-1}\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{e}_0\right)\right] \\ \mathbf{ct}_{\mathfrak{m}} = \left[\mathsf{ct}_{\mathfrak{m},\ell_{\mathfrak{m}}-1}, \cdots, \mathsf{ct}_{\mathfrak{m},0}\right] \in \left[\mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}\ell_{\mathfrak{m}}-1\right), ..., \mathsf{LWE}_{\boldsymbol{s}}\left(\mathfrak{m}_0\right)\right] \end{cases} \end{cases}$

1 $\mathsf{ct}_1 = \left[\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{m}_1}, \mathsf{ct}_{\mathfrak{s}_1}\right] \leftarrow \mathsf{TrivialEncryptFloat}\left(1\right)$

2 $\mathsf{ct}_{-1} = \left[\mathbf{ct}_{\mathfrak{e}_{-1}}, \mathbf{ct}_{\mathfrak{m}_{-1}}, \mathsf{ct}_{\mathfrak{s}_{-1}}\right] \leftarrow \mathsf{TrivialEncryptFloat}\left(-1\right)$

3 $\overline{\overline{\mathsf{CT}}}_{\mathfrak{s}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{\mathfrak{s}}, \mathsf{PUB}\right)$; `/* encrypt 0 if` $sign == 0$`, 1 otherwise */`

4 $\mathsf{ct}_{\mathfrak{s}_{\mathsf{tmp}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathsf{ct}_{\mathfrak{s}_1}, \mathsf{ct}_{\mathfrak{s}_{-1}}, \overline{\overline{\mathsf{CT}}}_{\mathfrak{s}}\right)$,
$\mathbf{ct}_{\mathfrak{e}_{\mathsf{tmp}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}_1}, \mathbf{ct}_{\mathfrak{e}_{-1}}, \overline{\overline{\mathsf{CT}}}_{\mathfrak{s}}\right)$,
$\mathbf{ct}_{\mathfrak{m}_{\mathsf{tmp}}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}_1}, \mathbf{ct}_{\mathfrak{m}_{-1}}, \overline{\overline{\mathsf{CT}}}_{\mathfrak{s}}\right)$

`/* If` $\mathsf{ct}_{\mathfrak{e}_{\ell_{\mathfrak{e}}-1}} \in \mathsf{LWE}_{\boldsymbol{s}}(x)$ `with` $x < 2^{\rho_{\mathfrak{e}}-1}$ `then` $\mathsf{LWE}_{\boldsymbol{s}}(1)$ `else` $\mathsf{LWE}_{\boldsymbol{s}}(0)$ `*/`

5 $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}} \leftarrow \mathsf{KS\text{-}PBS}\left(\mathsf{ct}_{\mathfrak{e}_{\ell_{\mathfrak{e}}-1}}, \mathsf{PUB}, \mathsf{LUT}_{\mathfrak{e}}\right)$

`/* If` $\mathsf{ct}_{\mathfrak{m}_{\ell_{\mathfrak{m}}-1}} \in \mathsf{LWE}_{\boldsymbol{s}}(x)$ `with` $x < 2^{\rho_{\mathfrak{m}}-1}$ `then` $\mathsf{LWE}_{\boldsymbol{s}}(1)$ `else` $\mathsf{LWE}_{\boldsymbol{s}}(0)$ `*/`

6 $\mathsf{ct}_{\mathsf{tmp}_{\mathfrak{m}}} \leftarrow \mathsf{KS\text{-}PBS}\left(\mathsf{ct}_{\mathfrak{m}_{\ell_{\mathfrak{m}}-1}}, \mathsf{PUB}, \mathsf{LUT}_{\mathfrak{m}}\right)$

7 $\mathsf{ct}_{\mathsf{tmp}} \leftarrow \mathsf{ct}_{\mathsf{tmp}_{\mathfrak{m}}} + \mathsf{ct}_{\mathsf{tmp}_{\mathfrak{e}}}$; `/* tmp equals zero only if` $|\mathbf{Dec}(\mathbf{ct}_{\mathfrak{f}})| > 1$ `*/`
`/* encrypt 0 if tmp` $== 0$`, 1 otherwise */`

8 $\overline{\overline{\mathsf{CT}}}_{\mathsf{tmp}} \leftarrow \mathsf{CBS}\left(\mathsf{ct}_{\mathsf{tmp}}, \mathsf{PUB}\right)$

9 $\mathsf{ct}_{\mathfrak{s}} \leftarrow \mathsf{ExtendedCMux}\left(\mathsf{ct}_{\mathfrak{s}}, \mathsf{ct}_{\mathfrak{s}_{\mathsf{tmp}}}, \overline{\overline{\mathsf{CT}}}_{\mathsf{tmp}}\right)$,
$\mathbf{ct}_{\mathfrak{e}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{e}}, \mathbf{ct}_{\mathfrak{e}_{\mathsf{tmp}}}, \overline{\overline{\mathsf{CT}}}_{\mathsf{tmp}}\right)$,
$\mathbf{ct}_{\mathfrak{m}} \leftarrow \mathsf{ExtendedCMux}\left(\mathbf{ct}_{\mathfrak{m}}, \mathbf{ct}_{\mathfrak{m}_{\mathsf{tmp}}}, \overline{\overline{\mathsf{CT}}}_{\mathsf{tmp}}\right)$

10 **return** $\mathbf{ct}_{\mathfrak{f}} = \left[\mathsf{ct}_{\mathfrak{s}}, \mathbf{ct}_{\mathfrak{m}}, \mathbf{ct}_{\mathfrak{e}}\right]$

Future works

In future works, we aim to integrate the strengths of both methodologies presented in this paper: the minifloat and the HFP approaches. As detailed in Section 3.1, the WoP-PBS faces limitations when evaluating bivariate functions due to the exponential growth of the LUT. However, this constraint is less significant when WoP-PBS is used solely for computing univariate functions. Therefore, for intermediate precision (around 16 bits), we could adopt the representation proposed in Section 3.2, utilizing all algorithms introduced for HFP in combination with WoP-PBS where it proves efficient.

Additionally, recent improvements to the CBS in [WWL+24] should immediately enhance the running time of the algorithms introduced in this paper.

Although there is still a long way to go before achieving a fully practical deployment, this work narrows the gap between plaintext and homomorphic floating-point computations. This could open new avenues for implementing confidential machine learning algorithms, which often require a wide range of values.

References

[AN16] Seiko Arita and Shota Nakasato. Fully homomorphic encryption for point numbers. In *International Conference on Information Security and Cryptology*, pages 253–270. Springer, 2016.

[BBB+22] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization & larger precision for (T)FHE. *IACR Cryptol. ePrint Arch.*, page 704, 2022.

[BCL+23] Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, Adeline Roux-Langlois, and Samuel Tap. Faster secret keys for (t) fhe. *Cryptology ePrint Archive*, 2023.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder VL Pereira, and Nigel P Smart. Final: Faster fhe instantiated with ntru and lwe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 188–215. Springer, 2022.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. *IACR Cryptology ePrint Archive*, 2012:78, 2012.

[BST20] Florian Bourse, Olivier Sanders, and Jacques Traoré. Improved secure integer comparison via homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 391–416. Springer, 2020.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.

[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 *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 409–437, 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 *International Conference on the Theory and Application of Cryptology and Information Security*, pages 670–699. Springer, 2021.

[CSVW16] Anamaria Costache, Nigel P Smart, Srinivas Vivek, and Adrian Waller. Fixed-point arithmetic in she schemes. In *International Conference on Selected Areas in Cryptography*, pages 401–422. Springer, 2016.

[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.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, pages 617–640, 2015.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):229–253, 2021.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *IACR Cryptology ePrint Archive*, 2013:340, 2013.

[Hwa24] Vincent Hwang. Formal verification of emulated floating-point arithmetic in falcon. In *International Workshop on Security*, pages 125–141. Springer, 2024.

[Ins08] Institute of Electrical and Electronics Engineers. Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

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

[Lai17] Kim Laine. Simple encrypted arithmetic library 2.3. 1. *Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf*, 2017.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient

fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 227–256. Springer, 2023.

[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.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[LS22] Seunghwan Lee and Dong-Joon Shin. Overflow-detectable floating-point fully homomorphic encryption. *Cryptology ePrint Archive*, 2022.

[MBDD+18] Jean-Michel Muller, Nicolas Brisebarre, Florent De Dinechin, Claude-Pierre Jeannerod, Vincent Lefevre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, Serge Torres, et al. *Handbook of floating-point arithmetic*. Springer, 2018.

[ML20] Subin Moon and Younho Lee. An efficient encrypted floating-point representation using heaan and tfhe. *Security and Communication Networks*, 2020, 2020.

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. *Post-Quantum Cryptography Project of NIST*, 2020.

[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.

[WK19] Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 4, 2019.

[WWL+24] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. Circuit bootstrapping: faster and smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 342–372. Springer, 2024.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.