REFHE: Fully Homomorphic ALU

Zvika Brakerski[2], Offir Friedman[1], Daniel Golan[1], Alon Gurni[1], Dolev Mutzari[1], and Ohad Sheinfeld[1]

[1] dWallet Labs, `research@dwalletlabs.com`
[2] Advisor to dWallet Labs

Abstract. We present a fully homomorphic encryption scheme which natively supports arithmetic and logical operations over large "machine words", namely plaintexts of the form $\mathbb{Z}_{2^n}$ (e.g. $n = 64$). Our scheme builds on the well-known BGV framework, but deviates in the selection of number field and in the encoding of messages. This allows us to support large message spaces with only modest effect on the noise growth.
Arithmetic operations (modulo 2^n) are supported natively similarly to BGV-style FHE schemes, and we present an efficient bootstrapping procedure for our scheme. Our bootstrapping algorithm has the feature that along the way it decomposes our machine word into bits, so that during bootstrapping it is possible to perform logical operations (essentially addressing each bit in the message independently). This means that during a single bootstrapping cycle we can perform logical operations on n bits. For example, a "greater than" operation (if $x > y$ output 1, otherwise 0), only requires a single subtraction and a single bootstrapping cycle.
Along the way we present a number of new tools and techniques, such as a generalization of the BGV modulus switching to a setting where the plaintext and ciphertext moduli are ideals (and not numbers).

1 Introduction

Fully homomorphic encryption (FHE) [RAD+78, Gen09a] allows us to compute on encrypted data without decrypting it first and without any knowledge of the secret key. This makes it a prominent privacy enhancing technology with numerous potential applications [GH19, CJP21a]. Whereas there is ample motivation for using FHE in the real world, its utilization in practice is hindered by the overhead it incurs in computation and in storage. More explicitly, the resources, say in running time, required to run a computation over encrypted data may be orders of magnitude longer than running the same computation in the clear. Narrowing this gap to a tolerable level has been a major research direction of the FHE community for over a decade, and indeed some success has recently been reported, notably the recent announcement of the integration of Swift FHE into Apple devices as a part of iOS 18 [BTR24]. However the overhead remains prohibitive (or at least quite restrictive) for many desirable applications.

There are two main paradigms in the literature for practically-oriented FHE candidates. Both rely on structure related to the Learning with Errors (LWE) [Reg05] and Ring Learning with Errors (RLWE) [LPR13a], as put forth in

[BV11b,BV11a,GSW13]. The first is a more direct extension [BV11b,BV11a] and is utilized in the BGV, B/FV, CKKS schemes [BGV12, Bra12, FV12, CKKS17] and their successors, and is of a more *arithmetic* nature. We therefore refer to these as *arithmetic schemes*. Software libraries that take this approach include [HS20, SEA11, CKKS16]. The second follows the paradigm of [GSW13, DM14, CGGI20] and is more native for *boolean* operations, and we therefore refer to them as *boolean schemes*. Software libraries that take this approach include [Zam21, tfh17]. Let us discuss these approaches in slightly more detail below.

Arithmetic Schemes. These schemes natively support arithmetic operations over some *plaintext modulus* p. In BGV and B/FV the operations are carried out exactly over a discrete ring, whereas in CKKS, they are real valued and noisy, but they are all natively arithmetic. When implemented naively, this approach leads to exorbitant information overhead. Namely to very large ciphertexts that encrypt a small amount of data. This is partly mitigated by the use of *batching*: the ability to encrypt multiple plaintexts in parallel "slots" in the same ciphertext, and apply operations on them in parallel. Batching reduces the *amortized* overhead, both in terms of storage and in terms of computation, since an operation on the ciphertext corresponds to operations on many plaintexts.

Importantly, batching uses algebraic properties of the RLWE problem, and in particular requires a specific relation between p and the number field over which the scheme is defined. In particular this means that the strongest form of batching requires p which is an odd prime with some specific structure. There are techniques in the literature that allow to trade off the "amount of batching" with the form of the modulus [GHS12, HS21]. These techniques use field extensions to produce plaintext spaces of the form $\mathrm{GF}(p^d)$. Thus it is possible to batch finite-field elements of size, e.g., 2^{64}, but it is not possible to properly batch plaintexts from the ring $\mathbb{Z}_{2^{64}}$. We are not aware of solutions that allow batching for message spaces such as those targeted in this work, namely $\mathbb{Z}_p$ for a large p, specifically for p being a power of 2.

Either way, batching requires both aggregation of a large amount of data per ciphertext *and* the ability to split this large amount of data into small amounts (to fit in each slot) in such a way that per-slot homomorphic operations remain useful. This is simply not the case when we wish to work with large integers, which is how many modern computer systems operate.

As in all known approaches for FHE, if one wishes to compute functionalities with a-priori unbounded depth, then a *bootstrapping* operation needs to be executed periodically. Bootstrapping [Gen09b] reduces the "noise component" that exists in all known FHE candidates. The noise grows with every homomorphic operation, and must be kept below a certain threshold to maintain the correctness of the scheme. Bootstrapping, therefore, is the process that allows to reduce the noise level so that additional homomorphic operations can be carried out. Bootstrapping essentially requires to apply the decryption algorithm homomorphically over the ciphertext. This requires applying operations that are not "natively" arithmetic, such as rounding or truncation of least significant bits. Therefore, whereas bootstrapping is heavy on resources in all known FHE instantiations, arithmetic schemes generally struggle with bootstrapping more than boolean ones. This is likewise the case for any non-arithmetic operation, in particular boolean operations like "greater than" are challenging to implement.

Boolean Schemes. In these schemes, each ciphertext essentially encrypts a single bit. It is possible to pack multiple ciphertexts into a more compact representation, and various optimizations have been introduced in [DM14, CGGI20], but homomorphic operations are still performed at the bit level. This allows to perform logical operations natively. A particularly successful approach is taken by the TFHE scheme [CGGI20]. This proposed a way to perform the bootstrapping operation very efficiently, and therefore elect to perform bootstrapping after (or rather, as a part of) every operation. This framework makes it harder to work with data types that are naturally composed of multiple bits, such as number modulo p. In boolean schemes, one needs to represent large numbers as a sequence of bits and apply the appropriate boolean circuits to perform operations like addition and multiplication. This is possible, and indeed works naturally even for moduli of the form 2^n, but requires a large number of bootstrapping operations even for the simplest of operations (e.g. adding two n bit numbers as integers modulo n). We note that boolean schemes are very convenient for logical operations, e.g. "if" statements. For example, boolean comparison between two numbers, i.e. the predicate $a \geq b$, can be computed as easily as the difference $a - b$. Note that this is not the case for arithmetic schemes.

The Challenge: Arithmetic-Logic Unit for Machine Words. Our goal when using FHE is to be able to take existing code and convert it into running homomorphically with as little change as possible. Indeed, our algorithms and data types should remain oblivious to our choice of privacy preserving technique. We may therefore put forth the task of constructing a homomorphic framework that natively supports the "standard" operations of computer programs. In particular, this includes arithmetics over integers modulo 2^n, since integer data types in most architectures are of this form. At the same time we wish to be able to execute conditional statements, and apply logical operations on the bits of our numbers.[3] Indeed, the "slogan" for our goal is to implement an "Arithmetic Logic Unit" (ALU) inspired by such components that exist in CPUs. Such a unit would support arithmetics modulo 2^n, as well as logical operations at the bit level.

We note that both aforementioned approaches are universal (a.k.a Turing complete) and therefore allow in principle to implement an ALU. However, when concrete efficiency is concerned, if the native representation is only arithmetic or only boolean, then a change of representation will be required for some operations, which is usually prohibitive in terms of resources. This work therefore focuses on the following question.

Is it possible to design an FHE scheme that natively supports arithmetics modulo $\mathbb{Z}_{2^n}$ and supports boolean operations as well?

[3] We are not considering float-point operations at this point.

1.1 Our Contribution – FHE for Arithmetic and Logical Operations

We present a number of novel ideas that allow us to break from the existing paradigm and present a fully homomorphic encryption scheme with new capabilities. We first show how to construct a BGV-style scheme with native support for arithmetics over $\mathbb{Z}_{2^n}$ while not incurring the penalty that is usually associated with such plaintexts in "legacy" BGV. We then present a bootstrapping algorithm for our scheme which not only performs the "standard" bootstrapping features of reducing the noise in the ciphertext, but also, along them way, provides us access to the n individual bits in the binary representation of our $\mathbb{Z}_{2^n}$ message. This allows us, in the course of bootstrapping, to perform logical operations natively. Thus achieving both arithmetic and boolean logic without additional overhead.

For the first part, we construct an encryption scheme whose native plaintext space is $\mathbb{Z}_{2^n}$ (where n is a parameter). We show that arithmetic homomorphic operations (addition, multiplication) over this plaintext space can be performed essentially the same as in BGV-style FHE. However, the parameters of the scheme and noise scaling are comparable to BGV with plaintext $\{0, 1\}$. This is a crucial difference since prior to this work, a plaintext space of $\mathbb{Z}_{2^n}$ meant an additional factor of 2^{nd} in the noise, when evaluating a depth d arithmetic circuit. In contrast, in our scheme the growth would be roughly $n^{O(d)}$, which as explained above is comparable to binary plaintext. This has a cascading effect on all parameters of the scheme, since mild noise growth allows to reduce the so-called "noise ratio" of the underlying algebraic LWE problem, which in turn upgrades the security level of the scheme, which then allows to reduce the size of other parameters and maintain the same security level as previous schemes. As a result, we can instantiate our scheme with very large plaintext spaces that were previously prohibitive, with only minimal loss in performance. We note that we did not explore the possibility of batching in our scheme, so we only consider the setting of a single message per ciphertext. However, it may be possible to incorporate batching into our framework as well.

This is achieved by examining the properties of the BGV scheme in the algebraic setting (i.e. when it is based on structures stemming from the RLWE assumption and similar ones). We notice that, under the hood, BGV encryption can seemlessly be made to support any message space that is defined by an *ideal* in the ring where the scheme is defined (the *ring of integers* of some *number field*, or a subring thereof). This property was already noticed in the context of the NTRU scheme by Hoffstein and Silverman [HS00], and was used in the context of FHE in a number of works, e.g. [CLPX18, BCIV20]. However, these works fell short of achieving the goal of naturally supporting $\mathbb{Z}_{2^n}$ since they insisted on sticking with the use of *cyclotomic number fields* which has indeed been prevailing in the FHE literature. We deviate from this paradigm and show that working with non-cyclotomic number fields opens the door for great versatility in the design of the message space. We view this as a major contribution of this work. Indeed, this modification requires us to use algebraic variants of LWE that are defined over such number fields. Whereas this is not as widely used in the literature, we notice that to the best of our knowledge there is no reason to speculate that cyclotomic number fields would lead to more secure constructions (in fact, some argue that the opposite is more likely). We provide a detailed discussion on the security of algebraic LWE in our context.

As an additional contribution, we discuss the possibility of creating "Double CRT" encoding of the ciphertexts in our scheme. Double CRT [HS20] is a way to encode ciphertexts that relies on the decomposition of the ciphertext modulus into prime ideals of a cyclotomic ring. This implies a representation of a large ciphertext as an array of fairly-small numbers, where addition and multiplication in the ring translates into pointwise addition and multiplication of the elements of the array. Whereas on the face of it we may not use Double CRT in our scheme, since we do not know how to decompose ordinary integers into prime ideals in our ring, we show that it is possible to take the opposite approach and construct the ciphertext modulus as a product of "simple" ideals. This would indeed allow for decomposition, but would imply that the ciphertexts in our scheme are no longer naturally represented as vectors of polynomials, where each coefficient is in $\mathbb{Z}_q$ for some integer q, but instead are cosets of some ideal in our ring. We show that nevertheless it is possible to apply homomorphic evaluation in this case.

In the second part we wish to devise a bootstrapping algorithm for our scheme. The bootstrapping process is computationally labor-intensive since it requires evaluating a pretty complex function. Furthermore, the noise accumulation *during bootstrapping* directly reduces the homomorphic capacity of the scheme after bootstrapping. Therefore, there is a lot of effort in the literature in order to reduce the complexity of bootstrapping in various FHE schemes [GHS12,HS21,DM14,CGGI20,KDE$^+$24]. In particular, to come up with "decryption circuits" that require the least amount of resources and have the least noise accumulation.

In our scheme, we manage to introduce a decryption functionality which is both relatively mild in terms of resources, and allows us to extract the individual bits of the message in the course of bootstrapping. This means that during bootstrapping we can perform bit-level operations on the encrypted message, which we leverage to obtain logical/boolean evaluation capacity for our scheme.

Our starting point is the method of [KDE$^+$24] who proposed to bootstrap BGV-style ciphertext by reducing the task to that of bootstrapping non-algebraic ciphertexts (i.e. ones that are not defined over a ring). In fact, this is quite straightforward in BGV-style encryption, since one can decompose an algebraic ciphertext into a collection of non-algebraic ciphertexts simply by thinking about ring elements as polynomials, and considering one coefficient at a time (see technical overview below for additional details). They then use the [CGGI20,CJP21a] bootstrapping approach which has been designed for non-algebraic ciphertexts, and use it essentially as a building block.

We cannot follow this blueprint as is, since our algebraic ciphertexts do not decompose well in terms of coefficients. That is, our use of a general ideal to define the ciphertext space means that noise in our ciphertext is not added per-coefficient as in previous FHE schemes. We therefore design a novel approach for recursive decomposition of our algebraic ciphertext into a collection of non-algebraic ciphertexts. Essentially this works by noticing that we can extract a non-algebraic ciphertext encrypting the least significant bit. We then bootstrap this ciphertext and show how to use the bootstrapped ciphertext in order to produce a non-algebraic encryption of the next bit of the message. At the end of the process, we have bootstrapped versions of non-algebraic encryptions of all bits of the message. This allows us to perform many logical operations "for free": e.g. apply any permutation or shift on the bits, remove some of the bits or XOR them with each other. We can also apply more sophisticated operations such as bitwise AND using a bit more work.

We refer the reader to the technical overview below for a more detailed explanation on how our scheme works.

Finally, we consider concrete parameters for our scheme, with a plaintext space of $n = 64$ bits. We implemented our scheme and report implementation-specific details.

1.2 Other Related Works

This work addresses FHE in the circuit model. This means that we consider computation that is represented in a combinatorial form as a circuit with boolean or arithmetic gates. Until recently, FHE was only known to be applicable in this model. Recently, Lin, Mook and Wichs [LMW23] introduced the first FHE scheme that operates in the RAM model (with suitable preprocessing). Whether our techniques have implications on RAM-FHE remains a subject for future inquiry.

Following the preparation of this manuscript and prior to its public release, other independent works on related topics appeared. Prior to our work, it was not known how to bootstrap [CLPX18]-type FHE. Indeed, standard methods for bootstrapping BGV/BFV do not seem to be applicable for such schemes. We present a way to resolve this issue, and a different bootstrapping method was shown in [GV24]. We note that our method, when used within our framework, enables the additional functionality of binary operations for free. Another recently published work by Boneh and Kim [BK25] presents a way to homomorphically manipulate large plaintext spaces, including modulo powers of two. Their method is quite different from ours and uses nested Residue Number System (RNS) representation on top of the CKKS FHE scheme. Whereas their method appears superior in terms of throughput, we believe that our method could be preferable in terms of latency. In particular, our scheme can be used as a leveled scheme and support a number of addition and multiplications without requiring bootstrapping, whereas in [BK25] bootstrapping is inherent in multiplication. Furthermore, even if bootstrapping is needed, which is a sequential process in our case, we may still be competitive with [BK25], e.g. for the plaintext space $\mathbb{Z}_{2^{256}}$.

1.3 Paper Organization

Due to space limitations, some of the technical details are deferred to the supplementary material. Section 2 contains a technical overview of our work. Section 3 contains preliminaries and definitions (some standard definitions are deferred to the supplementary material). The basic scheme is presented in Section 4. The arithmetic operations, which are an extension of the BGV technique to our setting, are deferred to the supplementary material (Section C). Our bootstrapping algorithm and the derived homomorphic boolean/logic operations are described in Section 5 (the details of the correctness and security analysis are deferred to supplementary material, in Section D). Implementation details and performance are discussed in Section 6.

Our analysis for using algebraic modulus for the sake of double CRT is provided in the supplementary material (Section F). A detailed discussion on the security of our hardness assumption in our ring is provided in the supplementary material (Section E).

2 Technical Overview

We start with a common blueprint for LWE-based encryption. The ciphertext is a (column) vector $\mathbf{c}$, say in $\mathbb{Z}_q^n$, and the secret key $\mathbf{s}$ is a (row) vector of the same dimension, say in $\mathbb{Z}^n$. The ciphertext is generated so that

$$\mathbf{s} \cdot \mathbf{c} = \mu + p\varepsilon \pmod{q} \ . \tag{1}$$

We refer to q is the "ciphertext modulus" and p is the "plaintext modulus", and we require that p, q are coprime. The encrypted message μ can be interpreted as an element in $\mathbb{Z}_p$ and the variable ε is the "noise". So long as $|\varepsilon| \ll q$, it is possible to recover m from the ciphertext.

In algebraic LWE variants, $\mathbb{Z}$ is replaced with some other ring. Particularly the "ring of integers" of a number field, or a full-rank subring thereof. A very common choice for such a ring is $\mathcal{R} = \mathbb{Z}[x]/\langle f(x) \rangle$, where f is an irreducible monic polynomial of degree n.[4] A prevalent choice for f is a *cyclotomic* polynomial, in particular of the form $f(x) = x^n + 1$ for n being a power of 2. The reader may keep this example in mind until we explain how we deviate from it in this work.

Given such a ring, we may analogously consider $\mathbf{c} \in \mathcal{R}_q$ (where $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$), $\mathbf{s} \in \mathcal{R}$ so that

$$\mathbf{s} \cdot \mathbf{c} = \mu + p\varepsilon \pmod{q\mathcal{R}} \ , \tag{2}$$

where $\varepsilon \in \mathcal{R}$ and the messages are now drawn from $\mathcal{R}_p$.[5] Since elements in $\mathcal{R}_q$ can be represented as degree $(n-1)$ polynomials, the term $\mu + p\varepsilon \pmod{q}$

[4] This ring coincides with the ring of integers of the number field defined by $f(x)$ in some useful special cases, but not always.

[5] In fact, algebraic variants of LWE (such as RLWE) were originally defined slightly differently, using the *dual ring*. This definition allowed to prove worst-case hardness

can be considered one coefficient at a time. In each such coefficient there is a message part that comes from $\mathbb{Z}_p$ and a noise part that comes from the respective coefficient of ε. We conclude that if all coefficients of ε are small compared to q, denote this $|\varepsilon| \ll q$, then $\mu \in \mathcal{R}_p$ can be fully recovered.[6]

As shown in [BV11b, BV11a, BGV12], the above scheme can be made homomorphic and support arithmetic operations over the plaintext ring $\mathcal{R}_p$. Namely, addition and multiplication. Each such operation incurs a penalty in the size of ε, most significantly during multiplication. Indeed, even in the most noise-efficient multiplication methods, a depth d multiplication incurs a penalty of roughly p^d in the noise (in addition to other penalties that are not directly dependent on p). Furthermore, arithmetics in $\mathcal{R}_p$ is often not what is actually required. Indeed, common data types take a form of $\mathbb{Z}_p$ and not of polynomials with $\mathbb{Z}_p$ coefficients that are multiplied modulo $f(x)$. One way to achieve homomorphism with respect to $\mathbb{Z}_p$ is to encode messages $\mathsf{m} \in \mathbb{Z}_p$ as polynomials $\mu \in \mathcal{R}_p$ by just setting the free coefficient of μ to be m, and keep all other coefficients at 0 (or in more abstract term use the fact that $\mathbb{Z}_p$ is a subring of $\mathcal{R}_p$). This indeed does the trick but has two main drawbacks. First, the information rate achieved is quite poor. An element in $\mathcal{R}_p$ can encode n elements from $\mathbb{Z}_p$ in terms of information content, but the homomorphic requirement forces us to lose a factor of n in utilizing this capacity. Second, if p is large, then the noise growth is completely prohibitive.

Prior works, starting with [SS10], proposed a way to *amortize* the information utility. They show that by properly selecting f and p, it is possible to set up the $\mathcal{R}$ so that $\mathcal{R}_p \cong \mathbb{Z}_p^n$, as rings, with pointwise addition and multiplication. This means that it is possible to pack n elements from $\mathbb{Z}_p$ and apply operations on them in parallel (see also [GHS12] and other followup works). However, this solution is limited to specific values of p, and to a setting where it is indeed possible to "collect" n messages into one ciphertext. This method had been extended to other types of plaintext spaces, specifically field or ring extensions, see [HS21]. We are not aware of solutions that allow batching for message spaces such as those targeted in this work, namely $\mathbb{Z}_p$ for a large p, and in particular $p = 2^\ell$ for some ℓ. However, using the existing methods, even if batching can be achieved, the noise blowup problem remains.

Using Ideal Plaintext Modulus. We notice, as others before [HS00, GC14, CLPX18], that p needs not be in $\mathbb{Z}$ in order for the scheme to work. Indeed, we may consider an ideal $\mathfrak{p}$ in the ring $\mathcal{R}$, and consider ciphertexts for which

$$\mathbf{s} \cdot \mathbf{c} = \mu + e \pmod{q\mathcal{R}} \ , \tag{3}$$

for the problem. However, it is possible to convert into the form presented here while paying a "penalty" in the size of the noise. See, e.g., [RSW18], and also Section E in this work.

[6] For the informed reader, we note that we work here with the so-called "coefficient embedding". Whereas in some cases it is useful to work with the "canonical embedding" we find that for the analysis of our scheme the coefficient embedding provides more convenience and flexibility.

where we are guaranteed that $e \in \mathfrak{p}$. For our purposes we consider $\mathfrak{p} = \langle x - 2 \rangle$, so we can think of $e = (x-2)\varepsilon$, where ε is as before. In this case, the plaintext μ is in the ring $\mathcal{R}_\mathfrak{p} = \mathcal{R}/\mathfrak{p}$. Importantly, in many cases there is a ring homomorphism $\mathcal{R}_\mathfrak{p} \cong \mathbb{Z}_p$ for $p = |f(2)|$ (this holds for any $f(x)$ that is of interest in this work). However, in a cyclotomic ring, such p can never be a power of 2. We therefore propose to use *non-cyclotomic* polynomials. In particular we consider polynomials of the form $f(x) = x^n - x + 2$.

We note while the FHE literature mostly uses cyclotomics, this is done for reasons of convenience and structure (e.g. having automorphisms that can be used for some functionalities). There is no evidence that working over cyclotomics makes algebraic LWE more secure. In fact, some claim that to the contrary, that the additional structure of cyclotomic rings makes them more risky in terms of security, and designers should opt for other polynomials [BCLvV16]. We refer the reader to Section E for a more detailed discussion.

We can now explain our scheme. Our message space is $\mathbb{Z}_{2^n}$ for some parameter n, and our ring is $\mathcal{R} = \mathbb{Z}[x]/\langle f(x) \rangle$, $f(x) = x^n - x + 2$. We require an encoding method to map an elements $\mathsf{m} \in \mathbb{Z}_{2^n}$ into $\mu \in \mathcal{R}_\mathfrak{p}$ and vice versa. It is important that $|\mu|$ is as small as possible, for reasons we explain below. This is in fact not very difficult as one can verify that one can map any m into μ with $\{0, 1\}$ coefficients by simply considering the binary representation of the number $\mathsf{m} \pmod{2^n}$. In the other direction, for any $\mu = \mu(x)$, we may consider $\mu(2) \pmod{2^n}$. This mapping indeed implement the aforementioned ring homomorphism. In addition it has the useful property that the bits of m are exactly the coefficients of μ. This is a consequence of using the non-cyclotomic polynomial and will be invaluable to us down the line.

Therefore, when we wish to encrypt a message m, we first encode it into a polynomial μ and then use the above scheme. The arithmetic homomorphic properties such as addition and multiplication work very similarly to previous schemes, but now, instead of p^d penalty for depth d multiplication as in [BGV12] and related schemes, the penalty scales roughly with $(\log p)^d$, which is a significant improvement. We will not go into the details here, but the reason, essentially, is that the noise growth factor depends on the maximal ℓ_1 norm of plaintexts in the space. Since we can encode m into μ with $\{0, 1\}$ coefficient, this factor is roughly $n = \log p$.

We remark that in the description so far, n plays a double role. Both as a dimension for the algebraic LWE problem and as a parameter that determines the plaintext space. If desired we can decouple these two roles by working with the so-called "module" version of the algebraic LWE problem. In this case $\mathbf{c} \in \mathcal{R}_q^r$ for some "rank" r (and likewise for $\mathbf{s}$). This means that the dimension for the LWE problem is $r \cdot n$, but the plaintext space is determined by n. This subtlety is not going to be of particular importance for this high level discussion.

Arithmetic Homomorphic Operations. Homomorphic addition and multiplication are performed similarly to the BGV scheme [BGV12, HS20]. Whereas many of the subroutines carry over, there is one that requires rethinking and updating. This is the so-called "modulus switching" procedure. The goal of this operation is essentially to "shrink down" the ciphertext modulus q into a new smaller q'. In this process, the relative noise level $|\varepsilon|/q$ remains roughly unchanged: $|\varepsilon|/q \approx |\varepsilon'|/q'$, which means that the absolute noise level goes down (since $q' < q$). This subroutine is quite central to BGV-style schemes and it needs to be carried out after every multiplication operation, and it also plays a role in the bootstrapping procedure (to be discussed below). Furthermore, [HS20] requires modulus switching in order to optimize the performance of another subroutine, called "key switching", which can in principle be carried out without modulus switching but with worse performance in terms of noise growth.

In this context, we first consider the straightforward extension of "legacy" modulus switching to the setting where $\mathfrak{p}$ is a general ideal. This is a relatively straightforward extension but it requires that $q = q' \pmod{\mathfrak{p}}$. Whereas prior to our work this was not considered a problem since essentially by definition, the plaintext space was such that $p \ll q, q'$, in our case this could be rather prohibitive. In fact, our paradigm even supports a setting where $q \ll p$. In prior works, [GC14] considered the straightforward extension as defined above, without noticing its drawbacks for very large p, whereas [CLPX18] did not face this challenge since they worked with the so-called B/FV approach, and do not perform bootstrapping, so at least asymptotically one can do without modulus switching, an approach that is not suitable for our needs.

We propose two methods to mitigate this problem. The first is to notice that even if $\alpha = qq'^{-1} \pmod{\mathfrak{p}} \neq 1$, so long as this value α is invertible modulo $\mathfrak{p}$, it is possible to "correct" the modulus switching procedure at the cost of an additional homomorphic scalar multiplication, which is relatively mild in terms of noise cost. The second is to do away with the ciphertext modulus q being an integer, and taking it as an element of the ring as well. This means that for any $\mathfrak{q}$ it is possible to find $\mathfrak{q}'$ with the proper "volume" (the volume in this context is the index of the ideal in the ring) so that modulus switching has the desired properties. We notice that working with such algebraic ciphertext modulus may have additional advantages. For example, setting $\mathfrak{q} = \prod_i \mathfrak{q}_i$, where each $\mathfrak{q}_i$ is "small" will allow to perform modulus switching as desired, and also provide a method for double-CRT encoding of the ciphertext, as explained above. We elaborate on this in Section F. Finally, going back to the first method, we notice that the scaling by α may be performed only "conceptually" without actually performing the multiplication. We can just "carry" the α factor throughout the computation and cancel it only at decryption (or at bootstrapping).

Bootstrapping. We now describe the bootstrapping procedure for our scheme. As explained above, we follow [KDE+24] and rely on the TFHE bootstrapping procedure [CGGI20] as a building block. Their goal was to bootstrap schemes with syntax similar to Eq. (2). They do this by decomposing the ciphertext into n non-algebraic ciphertexts, each of the form of Eq. (1). To see why this is the case, we recall that Eq. (2) is an equality between two ring elements, which can be seen as an equality between two polynomials of degree $n-1$. If we consider the i-th coefficient of this polynomial, then on the right hand side we get $\mu_i + p\varepsilon_i$, and on the left hand side we get a bilinear operation on the coefficients of $\mathbf{c}$

and $\mathbf{s}$. This can be interpreted as a non-algebraic ciphertext that is decryptable using the coefficient vector of $\mathbf{s}$. The [KDE+24] approach is to bootstrap each such non-algebraic ciphertext using the methods of [CGGI20], and then putting the outcomes back together.

This approach doesn't carry over to our setting. We embrace the basic idea of splitting an algebraic ciphertext into multiple non-algebraic ones, but we require a very different mechanism for this purpose, one that will ultimately also allow us to perform boolean operations in the course of bootstrapping. To see why this is the case, let us analyze the right hand side of Eq. (3) similarly to what we did with Eq. (2). The i-th coefficient of of $\mu + e$ equals $\mu_i + e_i$, but now recall that $e = (x-2)\varepsilon \pmod{f(x)}$, so e_i can no longer be written as $p\varepsilon_i$ as before. Let us write down the coefficients of e for $n = 4$ to get a sense of what is going on (note that ε_{n-1} wraps around because of the modulation in $f(x)$).

$$\begin{aligned}
e_0 &= -2\varepsilon_0 - 2\varepsilon_3 \\
e_1 &= \varepsilon_0 - 2\varepsilon_1 - \varepsilon_3 \\
e_2 &= \varepsilon_1 - 2\varepsilon_2 \\
e_3 &= \varepsilon_2 - 2\varepsilon_3
\end{aligned}$$

Note that other than e_0, in all other components there is no guarantee that the noise belongs to some ideal. Indeed the structure of the noise in our setting is not axis-parallel in the coefficient embedding. Nevertheless, we notice that at least for e_0 we do have a guarantee that it can be written as $e_0 = 2e'_0$. We may therefore extract the free coefficient from equation Eq. (3), and use it to bootstrap and obtain a non-algebraic low-noise encryption of μ_0.

It may seem that we have reached a dead end, since the other e_i are not as accommodating. To get us out of this barrier, let us consider a ciphertext for which we are guaranteed that $\mu_0 = 0$. Namely, it is possible to write $\mu = x\mu'$. We also notice that in our ring, since we work modulo $f(x) = x^n - x + 2$, it holds that $(x-2) = x^n$. It follows that we can write the right-hand side of Eq. (3), for such a ciphertext, as $x\mu' + x^n\varepsilon$. Using a standard technique (essentially multiplying the ciphertext by $x^{-1} \pmod{q}$) it is possible to remove the common factor and convert this to one whose right hand side is $\mu' + x^{n-1}\varepsilon$. Letting $e' = x^{n-1}\varepsilon$, we can see that $e'_0 = -2\varepsilon_1$, which is again even. In fact, for every power of x it holds that the free component of $x^i\varepsilon$ is even. This is not an accident and follows directly from our choice of ring polynomial $f(x)$. Our approach, therefore, is to first extract μ_0, and then subtract it from the original ciphertext to obtain a ciphertext as described above. Then extract μ_1 from this ciphertext and so on. However, implementing this approach is not without difficulties.

The first difficulty we encounter is that [CGGI20] expects a BFV-like ciphertext, where the right-hand side is of the form $\lceil q/2 \rfloor \mu_0 + \hat{\varepsilon}_0 \pmod{q}$. This is essential for the their bootstrapping to work. Essentially, the first step in [CGGI20] is to mod-switch the input ciphertext such that the ciphertext modulus is a power-of-two. This is needed for the correctness of the blind rotation step, as the ciphertext modulus of the input should align with the power-of-two cyclotomic ring degree under which the *test polynomial* is encrypted. We therefore first convert our extracted BGV LWE encryption of μ_0 to a BFV LWE encryption. Importantly, starting from a BFV-like encryption in our proposed scheme does not solve the above issue. In this case, the right-hand side of the encryption would be of the form $\lceil q/(x-2) \rfloor \mu(x) + \hat{\varepsilon}_0 \pmod{q, x^n - x + 2}$, and so no bit of the message μ can be extracted to begin with.

So far, we did not get into the question of what the output of the bootstrapping actually looks like. We only mentioned that it is a "non-algebraic low-noise encryption". The right-hand side of this encryption would normally be of the form $\lceil q/2 \rfloor \mu_0 + \hat{\varepsilon}_0 \pmod{q}$ for some small integer value $\hat{\varepsilon}_0 \in \mathbb{Z}$. How can we use it to cancel out μ_0 from an algebraic ciphertext? To do this, we utilize the versatility of the bootstrapping procedure of [CGGI20], which allows to produce non-algebraic ciphertexts with-right hand side $g(\mu_0) + \hat{\varepsilon}_0 \pmod{q}$, for any function g with $\mathbb{Z}_q$ values (in fact, it is even somewhat more versatile, but this suffices for us). We use it to recover all powers of B multiples of μ_0. That is $B^j \mu_0 + \hat{\varepsilon}_{0,j} \pmod{q}$. This allows us, via subset sum, to recover an encryption for any integer multiple of μ_0. Indeed, this is only for non-algebraic ciphertexts, but it can be extended to algebraic ciphertexts as well. We therefore recover an algebraic ciphertext of the form $(x-2)^{-1}\mu_0 + \hat{\varepsilon}'_0$, which can then be scaled to obtain a ciphertext of the form $\mu_0 + (x-2)\hat{\varepsilon}'_0$. This ciphertext can then be subtracted from the original algebraic encryption of μ, and allow us to continue to μ_1 and then further down the line.

To summarize, in each step of the bootstrapping, we recover an algebraic ciphertext of the form $x^i \mu_i + (x-2)\hat{\varepsilon}'_i$. We can eventually add all of these together to obtain an encryption of μ with small noise, which concludes the bootstrapping functionality.

Overall, even if the plaintext space is not all of $\mathbb{Z}_{2^n}$ but $\mathbb{Z}_{2^k}$ for $k \leq n$, the bootstrap complexity is dominated by $k \log_B(Q)$ [CGGI20] 'bit bootstraps', when the ciphertext modulus grows from q to Q, and the error grows by a polynomial factor of k, B and $\log(Q)$.

Logical Operations. As explained above, and demonstrated in our description of bootstrapping, in the course of bootstrapping we recover individual encryptions of all bits μ_i, scaled by any algebraic or non-algebraic integer. This allows us to very easily perform bitwise operations on the ciphertext. Bit shifts and permutations are essentially trivial since we can recover $x^i \mu_j + (x-2)\hat{\varepsilon}'_i$ for any i, j, and add them together to obtain the new ciphertext. Linear GF(2) operations over the bits of the message can likewise be computed.

More elaborate operations are also easily possible. For example the logical "$a \geq b$" predicate which outputs 1 if $a \geq b$ and 0 otherwise can be implemented homomorphically by computing (arithmetically) the difference $a - b \pmod{2^n}$ and then extract the most significant bit of the difference which exactly implements the above functionality.

We may even consider further operations such as bitwise AND between ciphertexts, which can also be performed once we extract all bits. One way to implement this is to recover non-algebraic ciphertexts of the form $\mu_i + 4\varepsilon_i$. Adding two ciphertexts of this form, and then extracting the second-least-significant bit, recovers the AND of the two numbers. One may think of more sophisticated algorithms that can be carried out in this way.

2.1 Estimated Performance

To conclude the technical overview, we provide an estimate of the performance of our scheme on its intended use-case, i.e. evaluating arithmetic circuits over a large ring $\mathbb{Z}_{2^k}$. Our benchmark for the comparison is the use of bit-by-bit TFHE encryption [CGGI20].[7] Since bootstrapping accounts for the vast majority of the computational cost of the homomorphic evaluation, we focus on a comparative analysis of the cost of bootstrapping.

We did not implement our bootstrapping procedure at this point, due to its complexity. However, since our bootstrapping procedure relies on the [CGGI20] bootstrapping procedure as a building block, we can readily compare the complexity by referring to the properties of this building block.

The Setting. We consider evaluating an arithmetic circuit over $\mathbb{Z}_{2^k}$, containing addition and multiplication operations. We stress that the parameter k needs not be identical to the REFHE ring degree n. One may think of the running example of $k = 64$ (whereas n will be much larger, as discussed below).

In the case of TFHE, the circuit is broken into boolean gates. Each addition operation will therefore require $\sim k$ gates, and each multiplication will require $\sim k^2$ gates. The application of each gate requires one application of the [CGGI20] bootstrapping procedure that we denote by Atomic. As described above, this operation takes as input a "BFV-like" LWE ciphertext with dimension n and modulus q. It produces a bootstrapped LWE ciphertext of dimension N and modulus Q.

We compare this with an instantiation of REFHE that supports multiplicative depth 2 before requiring to bootstrap. Therefore, the bootstrapping cost of REFHE in this case is $k \log_B(Q)$ applications of Atomic, potentially with *different* values for N, Q.

In terms of the *number* of Atomic calls, REFHE requires $k \log_B(Q)$ such calls, per two layers of evaluated circuit.[8] The number of calls required by TFHE depends on the structure of the circuit. At the very least, k^2 calls are required if the circuit just contains one multiplication gate.[9] We will use this very minimal setting for our comparison, but note that there are cases where the balance leans much more significantly in our favor. For example, when computing an inner product of d-dimensional vectors over the ring, an implementation using TFHE requires $dk^2 + (d-1)k$ calls to Atomic, whereas the REFHE cost remains $k \log_B(Q)$. For very large values of d, this can be very significant.

[7] We recall that our intended use-case is a setting where batching many plaintexts together is impractical or otherwise undesired. This leaves TFHE as a leading candidate for comparison.

[8] We will explain below that for our parameter selection, this number can be reduced, even to only k calls, but for now we keep the naive count $k \log_B(Q)$.

[9] Note that in the case where there is no multiplication, REFHE can evaluate without bootstrapping at all.

We will show that even if the circuit is set up to be least favorable for REFHE, we still expect to be competitive with the TFHE cost.

The Parameters of Atomic. Recall that Atomic takes a ciphertext with dimension n and modulus q, and produces an LWE ciphertext with dimension N and modulus Q. Under the hood, the output LWE ciphertext is extracted from a "Module LWE" ciphertext with module rank r and ring dimension $N/r := N^0$. The output ciphertext is expected to have "low noise". The measures for the performance of Atomic therefore are the computational complexity of execution, and the output noise level.

Since REFHE and TFHE call Atomic with different parameters, we need to explain the dependence of the complexity of Atomic on its various parameters. Using key switching and modulus switching, we can set up the REFHE bootstrapping so that the parameters of the input ciphertext to Atomic, namely n, q, are essentially identical to those in TFHE. The effect of these parameters is therefore transparent in our comparison.

For ease of comparison, we will also select our parameters so as to keep another important value invariant. Specifically, Atomic uses digit-decomposition gadget for the output modulus Q, with radix B'. Our value of Q will be much larger than that of TFHE, but we will increase B' accordingly so that $\log_{B'}(Q)$, i.e. the number of "digits" in Q represented in bases B', remains invariant between our parameters and theirs.

We are left with two parameters: the output modulus bit-length ratio: $\rho_Q = \frac{\log(Q_{\text{REFHE}})}{\log(Q_{\text{TFHE}})}$ (which, by the invariance of $\log_{B'}(Q)$, is also equal to $\frac{\log(B'_{\text{TFHE}})}{\log(B'_{\text{REFHE}})}$) and the output LWE dimension ratio: $\rho_N = \frac{N^0_{\text{REFHE}}(r_{\text{REFHE}}+1)}{N^0_{\text{TFHE}}(r_{\text{TFHE}}+1)}$. Therefore, calling Atomic with the parameters for REFHE will take $\times\rho_Q\rho_N$ more than for TFHE. That is the case because, using the double CRT representation [HS20], the cost of each arithmetic operation on ciphertexts is proportional to $N^0(r+1)\log Q$. The effect on the noise will be analyzed later.

Improved Performance by Amortization. This amortization uses additional lower-level properties of Atomic. In fact, Atomic takes its input ciphertext $\mathbf{c}$ and evaluates $\mu^* = \langle \mathbf{c}, \mathbf{s} \rangle \pmod{N_0}$. It can then evaluate an *arbitrary* function T with range in $\mathbb{Z}_Q$ on this value μ^* and produce a low-noise ciphertext $\mathbf{c}'$ modulo Q s.t. $\langle \mathbf{c}', \mathbf{s} \rangle \approx T(\mu^*)$. More details about the TFHE bootstrapping algorithm can be found in Appendix G. By looking at the details of the procedure, it is easy to see that one can also extract a ciphertext corresponding to $T(\mu^* + i)$ for any i that is known in advance. This means that we can take N_0 s.t. $q|N_0$. Let $u = N^0/q$. We can then run Atomic on $u \cdot \mathbf{c}$ rather than on $\mathbf{c}$ itself. This means that now T gets evaluated on $u \cdot \mu^*$. Therefore, if we have u different functions $T_0, \ldots, T_{u-1}$ that expect a modulo q input, we can define a function T with modulo N^0 input such that $T(u\mu^* + i) = T_i(\mu^*)$. This means that we can use Atomic to extract u different values computed on the same μ^* rather than just one. As a result, if $\frac{N^0_{\text{REFHE}}}{N^0_{\text{TFHE}}} \leq \log_B(Q)$, we need to call Atomic only $\log_B(Q) \cdot \frac{N^0_{\text{TFHE}}}{N^0_{\text{REFHE}}} \cdot k$ per bootstrapping, instead of $\log_B(Q) \cdot k$ times.

Parameter Selection. We set B' s.t. $\log_{B'}(Q)$ is invariant between our scheme and TFHE. We set B such that $\log_B(Q) = N^0_{\text{REFHE}}/N^0_{\text{TFHE}}$. This way we can use maximal amortization from the previous paragraph, so that our scheme only makes k calls to Atomic. The exact parameters are presented in Table 1.

Parameter	TFHE	REFHE
“Outer” radix B	NA	2^{20}
Ciphertext modulus Q	2^{32}	2^{160}
Inner radix B'	2^{10}	2^{50}
Gadget Levels	2	2
Module rank r	3 or 2	2 or 1
ring dimension N^0	512 or 1024	4096 or 8096
Noise after bootstrap	$\leq 2^{32}$	$\leq 2^{108}$
Remaining mult. depth	0	1
Ratios	$\rho_Q = 5$, $\rho_N = 6$ or $16/3$	
Cost of Atomic	τ_1 or τ_2 (reference values)	$30\tau_1$ or $\frac{80}{3}\tau_2$
Total Cost	$(k^2 \cdot \#_{\text{mult}} + k \cdot \#_{\text{add}}) \cdot \tau_i$	$\leq 30k\tau_i$

Table 1: Parameters in the bootstrapping procedure in TFHE and REFHE

Notice that the complexity ratio between $\mathsf{Atomic}_{\text{REFHE}}$ and $\mathsf{Atomic}_{\text{TFHE}}$ is between $27 - 30$.

We can now bound the post-bootstrapping noise in REFHE as follows. The noise is at most linear in $B' \cdot (N^0)^2$. The growth in N^0 and B' in REFHE might therefore increase the noise in Atomic by at most $2^8 \cdot 2^{40} = 2^{48}$ compared to TFHE. Choosing $B = 2^{20}$, we are left after bootstrapping with noise of at most $Q_{\text{TFHE}} \cdot \frac{(N^0_{\text{REFHE}})^2 \cdot B'_{\text{REFHE}}}{(N^0_{\text{TFHE}})^2 \cdot B'_{\text{TFHE}}} \cdot B = 2^{108}$.

Next, we show that using these parameters for REFHE, we can indeed support depth-1 homomorphic multiplication. To see this, observe that right before bootstrapping, the noise increases by at most a factor of 2^{15} due to the key-switching and modulus switching applied to ensure that the input ciphertext parameters closely match those in TFHE. For $Q = 2^{160}$, during homomorphic operations, the noise can increase by at most

$$\frac{Q}{2^{108} \cdot 2^{15}} = 2^{37}$$

while still preserving correctness. This bound is sufficient for multiplications of depth 1.

Conclusion. For $k = 64$, the complexity of multiplying two numbers in $\mathbb{Z}_{2^{64}}$ using TFHE (which is k^2) is, by our estimate, twice the cost of bootstrapping in our scheme, which in addition supports additions essentially ”for free”. We note that this is only an estimation of the dominant terms in the complexity of homomorphic evaluation, and actual runtime could depend on implementation details. We also emphasize that for computations over smaller plaintext spaces (e.g., 2^8), or those requiring many Boolean operations, the TFHE scheme appears to be preferable. Nonetheless, this comparison already shows that our scheme should at least be in the ballpark of TFHE in terms of performance, if not better. Furthermore, our suggested parameters here are preliminary and are not optimized. Indeed, even with this naive choice of parameters, our scheme appears to compete well with TFHE.

In terms of ciphertext size, which is of high importance, e.g. in blockchain applications, our ciphertexts are expected to be smaller by an order of magnitude or two compared to those in TFHE, as illustrated in Figure 2a (page 29).

3 Preliminaries

3.1 Notation

The security parameter is denoted by κ. We use bold letters for vectors and matrices.

An expression that includes a multiplication of a matrix and a ciphertext, or a vector of ciphertexts as in $D \cdot (c_0, \ldots, c_{n-1})^t$ is done as in the regular way, however when a matrix entry by a vector entry, the operation is a homomorphic scalar-by-ciphertext multiplication one.

3.2 Algebraic Number Theory

We consider a monic irreducible polynomial $f(x)$ of degree n and integer coefficients, the number field $K = \mathbb{Q}[x]/\langle f(x) \rangle$, and the ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x) \rangle$. Note that this ring is not necessarily the ring of integers of K, but it is nonetheless an order of the field (a full-rank subring of the ring of integers). For any ideal $\mathfrak{p}$ in $\mathcal{R}$ we denote $\mathcal{R}_\mathfrak{p} = \mathcal{R}/\mathfrak{p}$. For brevity, when $\mathfrak{q} = \langle q \rangle$ is generated by an integer $q \in \mathbb{Z}$, we denote $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. If $\mathfrak{p} = \langle x - a \rangle$ then there exists a ring-isomorphism $\mathcal{R}_\mathfrak{p} \cong \mathbb{Z}_p$ where $p = \mathfrak{N}(\mathfrak{p}) = |f(a)|$ (the notation $\mathfrak{N}(\cdot)$ refers to the *absolute norm* of the ideal which is defined as the index of the ideal in the ring: $\mathfrak{N}(\mathfrak{p}) = (\mathcal{R} : \mathfrak{p})$). In this work we will consider ideals for which the above holds. We are particularly interested in the setting where $f(x) = x^n - x + 2$ for some n, and $\mathfrak{p} = \langle x - 2 \rangle$. This results in $p = |f(2)| = 2^n$.

We consider two popular methods for *embedding* elements in K into Euclidean or complex space.

– **The Coefficient Embedding.** For a number field K as above, it is always possible to express any element of K uniquely as a polynomial of degree less than n and rational coefficients. This induces a map $K \to \mathbb{Q}^n$ by mapping $t \in K$ whose polynomial representation is $t = \sum_{i=0}^{n-1} t_i x^i$ to $[t] = (t_0, \ldots, t_{n-1})^t \in \mathbb{Q}^n$.
– **The Canonical Embedding.** A number field K has exactly n ring embeddings (injective ring homomorphisms) into $\mathbb{C}$, which are induced by taking x to the (complex) roots of f. Denote them by $\sigma_i : K \to \mathbb{C}$. The canonical embedding of t is the vector $[\![t]\!] = (\sigma_1(t), \ldots, \sigma_n(t))^t \in \mathbb{C}^n$. It is also possible

to embed this vector into $\mathbb{R}^n$ (essentially since f is a real-valued polynomial so its complex roots come in conjugate pairs). However, we will not require this real embedding here. In the canonical embedding, field addition and multiplication are done component-wise.

These embeddings induce geometric norms on elements of K by using ℓ_p norms over the embedding vectors. In this work we work mostly with the coefficient embedding (even though in some cases the canonical embedding would allow for a tighter analysis). We therefore let $\ell_p(\cdot)$ denote the ℓ_p norm of a field element in the coefficient embedding.

When working with the coefficient embedding, it is useful to consider the *expansion factor* of the field:

$$\gamma_p = \gamma_p(K) = \max\left\{\frac{\ell_p(a \cdot b)}{\ell_p(a)\ell_p(b)} : a, b \in K\right\} \tag{4}$$

By default, when p is not specified, we consider the infinity norm $p = \infty$.

For $w \in \mathcal{R}$ we define

$$\gamma_w = \max\left\{\frac{\ell_\infty(a \cdot w)}{\ell_\infty(a)\ell_\infty(w)} : a \in K\right\}$$

Our secret key has small ℓ_1 norm, therefore we also define:

$$\hat{\gamma} = \max\left\{\frac{\ell_\infty(a \cdot b)}{\ell_1(a)\ell_\infty(b)} : a, b \in K\right\} \ .$$

Note that $\hat{\gamma} \geq \gamma/n$. The following proposition bounds the expansion factor for our polynomial. The proof is via a direct calculation.

Proposition 3.1. *Let $f(x) = x^n - x + 2$, then for all $e_1, e_2 \in \mathcal{R}$ it holds that the expansion factor γ is upper bounded by $3n$, and moreover $\hat{\gamma}$ is upper bounded by 3.*

Proof. For $e \in \mathcal{R}$ we have:

$$\begin{aligned}(e_1 \cdot e_2)^{(i)} &= \sum_{j=0}^{i} e_1^{(j)} e_2^{(i-j)} - 2 \sum_{j=i+1}^{n} e_1^{(j)} e_2^{(n+i-j)} + \sum_{j=i}^{n} e_1^{(j)} e_2^{(n-1+i-j)} \\ &\leq 3 \cdot \ell_1(e_1) \cdot \ell_\infty(e_2) \\ &\leq 3n \cdot \ell_\infty(e_1) \cdot \ell_\infty(e_2) \ .\end{aligned}$$

3.3 Learning with Errors and Related Problems

The Learning with Errors (LWE) problem was introduced by Regev [Reg05] and became one of the most widely used and influential building blocks in cryptography. We use the following definition.

Definition 3.1 (Learning with Errors). *Let $n, q \in \mathbb{N}$, χ_s a distribution over $\mathbb{Z}^n$, χ_e a distribution over $\mathbb{Z}$.*

For a row vector $\mathbf{s} \in \mathbb{Z}^n$, consider the distribution $A_{\mathbf{s}}$ over $\mathbb{Z}_q^{n+1}$ defined as $\{(\mathbf{a}, \mathbf{sa} + e \pmod q)\}$, where $\mathbf{a}$ is uniform in $\mathbb{Z}_q^n$, e is sampled from χ_e.

Then the (decisional) Learning with Errors (LWE) problem with respect to parameters $\mathsf{lweparams} = (n, q, \chi_s, \chi_e)$ is to distinguish $A_{\mathbf{s}}$ from the uniform distribution over $\mathbb{Z}_q^{n+1}$, given an a-priori unbounded number of samples, where $\mathbf{s}$ is drawn from χ_s.

We refer to χ_s as the secret distribution and to χ_e as the noise distribution.

The Module LWE problem (MLWE) was introduced by the name "Generalized LWE" in [BGV12] as a generalization of the Ring LWE problem (RLWE) [LPR10, LPR13b]. Here we use a variant that analogous to the Polynomial LWE (PLWE) problem [RSW18], and defined as follows.[10]

Definition 3.2 (Module Polynomial LWE). *Let $f(x)$ be an irreducible monic polynomial of degree n, let $K = \mathbb{Q}[x]/\langle f(x)\rangle$ be a number field, define the ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x)\rangle$ and let $\mathfrak{q}$ be an ideal in this ring, denoting $\mathcal{R}_{\mathfrak{q}} = \mathcal{R}/\mathfrak{q}$. Let $r \in \mathbb{N}$, χ_s a distribution over $\mathcal{R}^r$ and χ_e a distribution over $\mathcal{R}$.*

For a row vector $\mathbf{s} \in \mathcal{R}^r$, consider the distribution $A_{\mathbf{s}}$ over $\mathcal{R}_{\mathfrak{q}}^{r+1}$ defined as $\{(\mathbf{a}, \mathbf{sa} + e \pmod{\mathfrak{q}})\}$, where $\mathbf{a}$ is uniform in $\mathcal{R}_{\mathfrak{q}}^r$, e is sampled from χ_e.

Then the (decisional) Module Polynomial LWE problem (MPLWE) with respect to parameters $\mathsf{mplweparams} = (f, n, r, \mathfrak{q}, \chi_s, \chi_e)$ is to distinguish $A_{\mathbf{s}}$ from the uniform distribution over $\mathcal{R}_{\mathfrak{q}}^{r+1}$, given an a-priori unbounded number of samples, where $\mathbf{s}$ is drawn from χ_s.

Remark 3.1. The special case of MPLWE where $r = 1$ is known as Polynomial LWE (PLWE) [SSTX09].

Remark 3.2. Letting $\chi_e = t \cdot \chi_\varepsilon$ for a ring element t which is coprime to q, it holds that MPLWE with noise sampled from χ_e is equivalent to the setting where the noise is sampled from χ_ε.

Remark 3.3. In this work, we use the term MPLWE to refer to the module polynomial LWE problem. We note that in prior work [RSSS17], the name MPLWE is used for the "middle-product" LWE problem. The latter is not directly related to our work.

Starting with [Reg05] there are numerous hardness results relating the hardness of LWE to the worst-case hardness of lattice problems, for ensembles of $\mathsf{lweparams}$ that range asymptotically with the security parameter. Similarly, there are worst-case hardness results for RLWE [LPR10, PRS17] and MLWE

[10] The difference between our variant in MLWE is the same as the difference between PLWE and RLWE. Whereas in RLWE, MLWE the secret, noise and arithmetics are done modulo the dual ring $\mathcal{R}^\vee$, in PLWE and MPLWE everything is defined over $\mathcal{R}$. Indeed one has to be careful when analyzing the security of such variants which we address in the appropriate section.

[BGV12, LS15]. There are also known methods for relating PLWE security to that of RLWE [RSW18]. These methods readily apply also to relate MPLWE security to MLWE. See Section E for a thorough discussion about the security of our assumptions.

4 Our Scheme

In this section, we introduce the encryption scheme, with the homomorphic properties being discussed in the subsequent section. As global parameters for our scheme, we consider the parameters $f(x) = x^n - x + 2, K = \mathbb{Q}[x]/\langle f(x)\rangle, \mathcal{R} = \mathbb{Z}[x]/\langle f(x)\rangle, \mathfrak{p} = \langle x-2\rangle, p = 2^n$, as defined in Section 3.2. We recall the ring isomorphism $\mathbb{Z}_p \cong \mathcal{R}_\mathfrak{p}$.

The ciphertext space is defined over $\mathcal{R}_q$ as defined in 3.2. Unless stated otherwise, additions and multiplications in the following sections are over $\mathcal{R}_q$. Given $a \in \mathcal{R}$, we denote by $[a]_{\mathcal{R}_q} \in \mathcal{R}$ the unique ring element with coefficients in the range $[-q/2, q/2)$ such that $[a]_{\mathcal{R}_q} = a \mod \langle q\rangle$.

In Section 4.1 we describe an encoding - decoding procedure for elements in $\mathbb{Z}_p$ into elements in $\mathcal{R}_\mathfrak{p}$. We proceed by introducing our encryption scheme in Section 4.2 and analyze correctness and security in Section B. Homomorphic properties are discussed in subsequent sections.

4.1 Messages vs. Plaintexts

Our scheme encrypts plaintexts μ which are elements in $\mathcal{R}_\mathfrak{p}$, i.e. as cosets of the ideal $\mathfrak{p}$. We use these plaintexts to encode messages $\mathsf{m} \in \mathbb{Z}_p$, where we recall that $p = (\mathcal{R} : \mathfrak{p})$.

We therefore require an efficient implementation of the ring isomorphism $\mathbb{Z}_p \cong \mathcal{R}_\mathfrak{p}$, in both directions, so that messages can be properly encoded and decoded. In terms of terminology, we differentiate between "*messages*" which are elements in $\mathbb{Z}_p$, and "*plaintexts*" which are elements in $\mathcal{R}_\mathfrak{p}$. Our encoding and decoding translate messages into plaintexts and vice versa.

We note that both $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ and $\mathcal{R}_\mathfrak{p} = \mathcal{R}/\mathfrak{p}$ are quotient rings, so we need to consider specific representatives that are produced by the encoding and decoding. We consider two procedures $\mathsf{Encode}, \mathsf{Decode}$ that run in polylog(p) time, take as input elements from $\mathbb{Z}, \mathcal{R}$ respectively, and output elements from $\mathcal{R}, \mathbb{Z}$, and implement the ring isomorphism. We further require that Encode produces "short" elements. Our measure of length in this context is infinity norm in the coefficient embedding. We let $B_{\mathsf{enc}} = \max_{\mathsf{m}\in\mathbb{Z}} \|\mathsf{Encode}(\mathsf{m})\|_\infty$.

Notably, in our scheme with $\mathfrak{p} = \langle x-2\rangle$, it is possible to achieve $B_{\mathsf{enc}} = 1$ since any coset of $\mathcal{R}_\mathfrak{p}$ has a representative with $\{0,1\}$ coefficients obtained by considering the binary representation of $\mathsf{m} \pmod p$ Namely, assume without loss of generality that $\mathsf{m} \in [0, p-1)$ and that $\mathsf{m} = \sum_i \mu_i 2^i$, where $\mu_i \in \{0,1\}$. Then $\mathsf{Encode}(\mathsf{m}) = \mu(x) = \sum_i \mu_i x^i$ is a valid encoding procedure whose output only has $\{0,1\}$ coefficients. For Decode, we notice that given $\mu = \sum_i \mu_i x^i$, we can output $\sum_i \mu_i 2^i \pmod p$.

We let $\mathcal{R}_{\{0,1\}}$ denote the set of plaintexts of our scheme. That is, $\mathcal{R}_{\{0,1\}}$ is the set of elements in $\mathcal{R}$ that can be represented as degree $(n-1)$ polynomials with $\{0,1\}$ coefficients.

We extend the $\mathsf{Encode}(\cdot)$ function to act also on elements from $\mathcal{R}$ by taking their small-coefficient representative modulo $\mathfrak{p}$. More formally, for $\mu \in \mathcal{R}$, we define $\mathsf{Encode}(\mu) = \mathsf{Encode}(\mathsf{Decode}(\mu)) \in \mathcal{R}_{\{0,1\}}$.

4.2 The Encryption Scheme

In this section we present our encryption scheme called *Ring Embedding FHE* (REFHE), based on the MPLWE hardness assumption (see Definition 3.2). We embed messages $\mathsf{m} \in \mathbb{Z}_p$ as ring elements $\mu = \mathsf{Encode}(\mathsf{m}) \in \mathcal{R}_{\{0,1\}}$, which results in a compact ciphertext.

Conventions ε is sampled form χ_e in $\mathcal{R}$. e is in $\mathfrak{p}$. We denote by B_{χ_ε} the bound on the ℓ_∞ of χ_ε and by B_{χ_e} the bound on the ℓ_∞ of $(x-2)\cdot\chi_\varepsilon$

Algorithm 4.1: The Encryption Scheme

1. **Setup**: $\mathsf{pp} \leftarrow$ REFHE.Setup$(1^n, 1^\kappa)$ gets n the degree of the polynomial, κ the security parmeter and Returns $\chi_s, \chi_\varepsilon, q, r$ as pp, such that $\gcd(q,2) = 1$, the MPLWE problem with $\mathsf{mplweparams}(f, n, r, q, \chi_s, \chi_\varepsilon)$ is κ-hard, and $q > q_0$ as defined in lemma B.2.
2. **Key Generation**: $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow$ REFHE.Keygen$(1^\kappa, \mathsf{pp})$. Sample row vectors $\mathbf{s}_1 \leftarrow \chi_s$, $\boldsymbol{\varepsilon} \leftarrow \chi_\varepsilon^r$, and a uniformly random matrix $\mathbf{A} \in \mathcal{R}_q^{r\times r}$. Returns the secret key $\mathbf{s} = (1, -\mathbf{s}_1)$, and public key $\mathsf{pk} = (\mathbf{b}, \mathbf{A})$ where $\mathbf{b} = \mathbf{s}_1\mathbf{A} + (x-2)\boldsymbol{\varepsilon}$. evk consists of the public parameters needed for homomorphic evaluations - relinearization key and bootstrap key, which we will describe through the paper.
3. **Encryption**: $\mathbf{c} \in \mathcal{R}_q^{r+1} \leftarrow$ REFHE.Enc$_{\mathsf{pk}}(\mu)$. For $\mu \in \mathcal{R}_{\{0,1\}}$, sample $\mathbf{r} \leftarrow \chi_s$, $\varepsilon_0 \leftarrow \chi_\varepsilon$, $\boldsymbol{\varepsilon}_1 \leftarrow \chi_\varepsilon^r$. Then $c_0 = \mathbf{b}\cdot\mathbf{r} + (x-2)\varepsilon_0 + \mu$, $\mathbf{c}_1 = \mathbf{A}\mathbf{r} + (x-2)\boldsymbol{\varepsilon}_1$. Return the column vector $\mathbf{c} = (c_0, \mathbf{c}_1)$.
4. **Decryption**: $\mu \leftarrow$ REFHE.Dec$_{\mathbf{sk}}(\mathbf{c})$ returns $[\mathbf{s}\cdot\mathbf{c}]_{\mathcal{R}_q} \mod \mathfrak{p} = [c_0 - \mathbf{s}_1\cdot\mathbf{c}_1]_{\mathcal{R}_q} \mod \mathfrak{p}$.

When the scheme is used as an homomorphic encryption scheme, the setup and key generation phases also depend on the homomorphic capacity, meaning in which circuit evaluation the scheme supports. In this case for the key generation also generates 'Key Switch matrices' as seen in Algorithm C.3. The decryption REFHE.Dec might be with respect to another modulus, as the modulus changes during the circuit evaluation due to the Key Switching and Modulus switching Algorithms C.3, C.5.

5 Bootstrapping and Boolean Operations

We now present our bootstrapping algorithm. We start by introducing a generic notation that will be used throughout this section. In the course of the bootstrapping we switch between a number of forms of algebraic and non-algebraic LWE. All of these schemes have a very similar syntax, namely a linear decryption over some ring results in an encoding of a plaintext with some noise. To capture this, we denote by $\mathrm{LWE}^{a,b}_{q,\mathbf{s}}(\mu;\varepsilon)$ the set of vectors of elements in $\mathbb{Z}_q$ such that for a secret key $\mathbf{s}$ it holds that

$$\mathbf{s}\cdot\mathbf{c} = b\mu + a\varepsilon \pmod q, \qquad a,b,\in\mathbb{Q}\ .$$

Intuitively $\mathrm{LWE}^{a,b}_{q,\mathbf{s}}(\mu;\varepsilon)$ can be thought of as a set of ciphertexts and related objects (i.e. modulus-switching parameters) generated by a scheme which is based on the LWE assumption. We therefore refer to such objects as "LWE ciphertexts".

Similarly, we denote by $\mathrm{MPLWE}^{a,b}_{q,\mathbf{s}}(\mu;\varepsilon)$ the set of vectors of elements in some polynomial ring $\mathcal{R}$ such that for a secret key $\mathbf{s}$ it holds that

$$\mathbf{s}\cdot\mathbf{c} = b\mu + a\varepsilon \pmod q, \qquad a,b\in K\ .$$

$\mathrm{MPLWE}^{a,b}_{q,\mathbf{s}}(\mu;\varepsilon)$ can be thought of as the analogous notion to $\mathrm{MPLWE}^{a,b}_{q,\mathbf{s}}(\mu;\varepsilon)$ but for schemes based on the MPLWE assumption. We therefore refer to such objects as "MPLWE ciphertexts".

We specialize the above notation to refer to objects that satisfy the syntactic requirements of specific schemes:

$$\mathrm{BGV}_{q,\mathbf{s}}(\mu;\varepsilon) = \mathrm{LWE}^{2,1}_{q,\mathbf{s}}(\mu;\varepsilon)$$
$$\mathrm{BFV}_{q,\mathbf{s}}(\mu;\varepsilon) = \mathrm{LWE}^{1,\frac{q}{2}}_{q,\mathbf{s}}(\mu;\varepsilon)$$
$$\mathrm{REFHE}_{q,\mathbf{s}}(\mu;\varepsilon) = \mathrm{MPLWE}^{(x-2),1}_{q,\mathbf{s}}(\mu;\varepsilon)$$

We sometimes omit one or more of the parameters $\mathbf{s}, a, b$ when they are clear from the context.

We use $q, Q \in \mathbb{N}$ to denote ciphertext space moduli. We use Q to denote the modulus of a newly encrypted REFHE ciphertext, and q to denote a modulus of a ciphertext that is the result of homomorphic operations and has potentially consumed ciphertext levels (and therefore underwent modulus switching) reaching to a level for which the modulus is q. It holds that $q \ll Q$.

The rest of this section is organized as follows. In Section 5.1 we cover the subroutines in more detail, and in Section 5.2 we present our bootstrapping algorithm. In Section 5.3 we show how boolean operations can be preformed homomorphically during bootstrapping.

5.1 Subroutines of the Bootstrapping Algorithm

5.1.1 Programmable Bootstrapping for B/FV This is the procedure from [KDE+24] (based on [CGGI20]) that allows to use the programmable bootstrapping of the TFHE scheme in order to bootstrap schemes that are based on the BGV and B/FV paradigms. Implicit in the bootstrapping procedure is a parameter mapping function $\mathsf{PMap}(n, q) \to (n', q')$ that maps the dimension and modulus of an incoming LWE ciphertext into the dimension and modulus of the bootstrapping parameters and ultimately the output ciphertext.

– Setup algorithm $\mathsf{PB.Setup}(1^n, 1^q, \mathbf{s}, \mathbf{s}')$, which is a randomized algorithm that takes as input the following values. LWE dimension and modulus parameters (n, q), both given in unary representation (note that this is contrary to standard conventions where the modulus is usually given in binary), and a corresponding binary LWE secret key $\mathbf{s} \in \{0, 1\}^n$. It also takes an LWE secret key s' of dimension n', where $(n', q') = \mathsf{PMap}(n, q)$. The algorithm outputs the programmable bootstrapping public parameters pbpp of bit length $\mathrm{poly}(n, q, n', \log q')$.
– Programmable Bootstrapping Algorithm $\mathsf{PB.Bootstrap}(\mathsf{pbpp}, \mathbf{c}, F)$ which is a deterministic algorithm that takes as input the programmable bootstrapping parameters pbpp an LWE ciphertext $\mathbf{c}$ with dimension n and modulus q, a function $F : \mathbb{Z}_q \to \mathbb{Z}_{q'}$ represented by its truth table. It outputs $\mathbf{c}'$, an LWE ciphertext with dimension n' and modulus q'.
  We overload the notation of $\mathsf{PB.Bootstrap}$ with respect to the last operand. Let F be a matrix of functions: $F \in (\mathbb{Z}_q \to \mathbb{Z}_{q'})^{m_1 \times m_2}$, then $\mathsf{PB.Bootstrap}(\mathsf{pbpp}, \mathbf{c}, F)$ is a shorthand for the procedure that applies $\mathsf{PB.Bootstrap}(\mathsf{pbpp}, \mathbf{c}, F_{i,j})$ for every entry of F and outputs a matrix $m_1 \times m_2$ of LWE ciphertexts corresponding to the outputs of the executions.

5.1.2 Sample Extract We next describe how to extract a LWE sample encrypting the free coefficient (recall that we face the constraint of being able to only use the free coefficient in our bootstrapping mechanism, the reason for which is explained in Section 2) of the input REFHE ciphertext as a BFV ciphertext. For a field element $t \in K$ we consider the operator $J : K \to \mathbb{Q}$ to be the operator that outputs the free coefficient $u \in \mathbb{Q}$, of the multiplication of t by a field element $w \in K$. Note that if we consider such a multiplication of elements in the ring $\mathcal{R}$ the result is in $\mathbb{Z}$ and hence, viewed as a linear operator acting on the coefficient embedding of $w \in \mathcal{R}$, its output is a coefficient vector in $\mathbb{Z}^n$. Extending J to operate on vectors of such ring elements $\mathcal{R}^r$ we take the output to be a concatenation of the resulting coefficient vectors, one for each of the ring elements, and denote it formally as $J : \mathcal{R}^r \to \mathbb{Z}^{nr}$.

The extraction of the first coefficient of a REFHE ciphertext $\mathbf{c} = (c_0, \mathbf{c}_1)$ encrypted under the secret key $\mathbf{s} = (1, -\mathbf{s}_1)$ is simply $([c_0]_0, J(\mathbf{c}_1))$, which will output an BGV ciphertext encrypting the same message under the extracted secret key, namely the coefficient embedding of $\mathbf{s}_1$.

After extraction we switch the output BGV ciphertext to a BFV ciphertext. This is done for two reasons: the first is that TFHE requires a modulus switching to one that is a power of 2 which is not possible for BGV since BGV requires q and p to be coprime, and the second is that during bootstrapping we evaluate a function that takes as input a ciphertext in which the noise term e is not multiplied by a coefficient as in BFV.

Starting with a BGV ciphertext $\mathbf{c}$, s.t. $\mathbf{s} \cdot \mathbf{c} = \mu + 2\varepsilon \mod q$, we switch it to a BFV ciphertext $\mathbf{c}'$ by multiplying by $2^{-1} \pmod q = (q+1)/2$, so we have:

$$\mathbf{c}' = \frac{q+1}{2}\mathbf{c}$$

and get:

$$\begin{aligned} \mathbf{s} \cdot \mathbf{c}' &= \frac{q+1}{2} \cdot \mu + \frac{q+1}{2} \cdot \varepsilon \mod q \\ &= \frac{q+1}{2} \cdot \mu + (q+1) \cdot \varepsilon \mod q \end{aligned}$$

Since we are performing modular arithmetic modulus q, we can omit the term $q \cdot \varepsilon$ and obtain:

$$\mathbf{s} \cdot \mathbf{c}' = \frac{q+1}{2} \cdot \mu + \varepsilon \mod q$$

which is indeed a BFV ciphertext.

Algorithm 5.1: Sample Extract

```
Input: $\mathbf{c} = (c_0, \mathbf{c}_1) \in \mathcal{R}_q^{r+1}$ a REFHE ciphertext encrypted under the
    secret key $\mathbf{s} = (1, -\mathbf{s}_1)$
Output: $c_{\mathrm{BFV}} \in \mathbb{Z}^{nr+1}$ a BFV ciphertext encrypting the first coefficient
    of $\mathbf{c}$ under the secret key $(1, -[\mathbf{s}_1])$ (recall that $[\mathbf{s}_1]$ is the coefficient
    vector of $\mathbf{s}_1$).
    $c' \leftarrow ([c_0]_0, J(\mathbf{c}_1))$
    return $c_{\mathrm{BFV}} \leftarrow \frac{q+1}{2} \cdot c'$
```

5.1.3 Repacking The aim of the Repacking procedure is to construct a MPLWE ciphertext of a degree n polynomial whose n coefficients are given under n corresponding LWE encryptions. This is done by converting each of them into a MPLWE ciphertext that encrypts the coefficient multiplied by the corresponding monomial x^i, and sum them up at the end. This is accomplished by applying a technique that resembles key-switching.

In what follows, we present the technique in more detail. We start with a vector of n LWE ciphertexts: $(\mathbf{c}_0, \ldots, \mathbf{c}_{n-1})$, such that:

$$\mathbf{s} \cdot \mathbf{c}_i = \alpha_i + \varepsilon_i \pmod Q$$

Next, we decompose each such ciphertext, and compute the vector $\tilde{\mathbf{c}}_i = \mathbf{g}^{-1}(\mathbf{c}_i) \in \{0,1\}^{n\lfloor \log_2 Q \rfloor}$, by applying the binary decomposition operator on each $\mathbf{c}_i$. Namely, $\tilde{c}_i^{jl}$ is the lth bit of the jth entry of $\mathbf{c}_i$. To this end, we use the key-switching keys (as described in Section 5.2.1). Recall that the key-switching key is a vector of MPLWE encryptions of $\mathbf{g}_{2,Q} \cdot \mathbf{s}$, that is, the powers of 2 gadget vector multiplied by the secret key elements, under some key denoted here as $\mathbf{s}'$. Specifically, the key switching keys are of the following form:

$$a_{j,l} + \mathbf{b}_{j,\ell}\mathbf{s}' = 2^l s_j + \hat{\varepsilon}(x) \qquad (\text{mod } Q, f)$$

We then compute the following subset sum (with respect to each $\tilde{\mathbf{c}}_i$) of the MPLWE key-switching ciphertexts $(a_{j,l}, \mathbf{b}_{j,l})$, and denote the resulting MPLWE ciphertexts by $(d_i[0], \mathbf{d}_i[1])$. Formally:

$$(d_i[0], \mathbf{d}_i[1]) = \mathbf{s} \cdot \mathbf{c}_i = \sum_{j,l} (a_{j,l}, \mathbf{b}_{j,l}) \cdot \tilde{c}_i^{jl}$$

By expanding the right-hand side, we get:

$$\begin{aligned} d_i[0] + \mathbf{d}_i[1]\mathbf{s}' &= \sum_{j,l} (a_{j,l} + \mathbf{b}_{j,l} \cdot \mathbf{s}')\tilde{c}_i^{jl} = \sum_{j,l} \tilde{c}_i^{jl}[2^l \cdot s_j + \hat{\varepsilon}_{j,l}(x)] \\ &= \alpha_i + \varepsilon_i + \sum_{j,l} \tilde{c}_i^{jl} \cdot \hat{\varepsilon}_{j,l}(x) \end{aligned}$$

Finally, denote by $d_0 = \sum_i d_i[0]x^i$, and by $\mathbf{d}_1 = \sum_i \mathbf{d}_i[1]x^i$ and observe that:

$$d_0 + \mathbf{d}_1\mathbf{s}' = \sum_i (d_i[0] + \mathbf{d}_i[1]\mathbf{s}')x^i = \alpha(x) + \varepsilon(x) + \sum_{i,j,l} \tilde{c}_i^{jl} \cdot \hat{\varepsilon}_{j,l}(x)x^i \quad \in \mathcal{R}_Q,$$

where $\alpha(x) := \sum_i \alpha_i x^i$, as desired.

Algorithm 5.2: Repacking

Input: $\mathsf{bspp.ksk} \in \mathcal{R}_Q^{r \times n\lfloor \log_2 Q \rfloor}$ key-switching keys as MPLWE ciphetexts
$\mathbf{c}$ a vector of n LWE ciphertexts

Output: a MPLWE ciphertext that encrypts the polynomial whose coefficients are encrypted under $\mathbf{c}$.

```
for $i = 0, \ldots, n-1$ do
    $\tilde{\mathbf{c}}_i \leftarrow \mathbf{g}^{-1}(\mathbf{c}_i)$
    $d_i = \sum_{j,l} \tilde{c}_i^{jl} \cdot \mathsf{ksk}_{j,l}$        ▷ $\tilde{c}_i^{jl}$ is the $l$th bit of the $j$th entry of $\mathbf{c}_i$
return $\sum_{i=0}^{n} d_i x^i$
```

5.2 The Bootstrapping Procedure

5.2.1 Bootstrapping Setup The procedure $\mathsf{REFHE.BS.Setup}$ defines public parameters bspp, which include: a function F_D, the function we evaluate during bootstrapping, gadget decomposition elements (as defined in Section A.2);

a decomposition base B used for the decomposition, gadget vector: $\mathbf{g} = \mathbf{g}_{B,Q} = (1, B^1, \ldots, B^{\lfloor \log_B Q \rfloor})$, gadget matrix $\mathbf{G} = \mathbf{G}_{\mathbf{g}} = \mathbf{I}_k \otimes \mathbf{g}$ and a gadget decomposition operation $\mathbf{G}^{-1} = \mathbf{G}_{\mathbf{g}}^{-1}$, key-switcing keys; for a secret key $\mathbf{s}$ generated by REFHE.Keygen as in Algorithm 4.1 in Section 4.2 a vector of key-switching keys consists of encryptions of the different powers of 2 multiplied by the bits of the secret key $\mathbf{s}$, namely a key-switching key $\mathbf{ksk}_{j,l}$) is in a REFHE encryption of $2^l \cdot \mathbf{s}_j$ where $0 \leq l < \lfloor \log_2 Q \rfloor$, TFHE bootstrapping parameters; pbpp output by the algorithm $\mathsf{PB.Bootstrap}$ as in Section 5.1.1.

5.2.2 Bootstrapping Algorithm

Definition 5.1. *For $q, Q \in \mathbb{N}$ we define $F_D : \mathbb{Z}_q \to \mathbb{Z}_Q$ to be the function that extracts the most significant bit of its input. Specifically $F_D(z)$ represents z as $(q/2)\mu + \varepsilon$ where ε has the smallest possible absolute value and $\mu \in \{0, 1\}$, and outputs μ.*

Algorithm 5.3: Bootstrapping

```
Input: $c_{\mathsf{IN}} = \text{REFHE}_q(\mu; \varepsilon)$    REFHE ciphertext to be bootstrapped,
       $\mathsf{bspp}$    bootstrapping parameters obtained at scheme setup,
Parameters: $F_D$ bootstrapping function, $\mathbf{g}$ gadget vector, $n$ security
    parameter, $q$ ciphertext modulus, $Q$ ciphertext modulus of freshly
    bootstrapped ciphertexts
Output: A REFHE ciphertext with dimension $n$ and modulus $Q$
 1: $c_{\ll} \leftarrow c' \leftarrow c_{\mathsf{IN}}$
 2: for $i = 0, \ldots, n-1$ do
 3:     $c_{\mathsf{lsb}} \leftarrow \mathsf{REFHE.SampleExtract}(c_{\ll})$        ▷ $c_{\mathsf{lsb}} = \text{BFV}_q(\mu_0; \varepsilon)$
 4:     $\tilde{\mathbf{c}} \leftarrow \mathsf{PB.Bootstrap}(\mathsf{pbpp}, c_{\mathsf{lsb}}, \mathbf{g} \cdot F_D)$      ▷ $F_D$ per Definition 5.1
 5:     $\hat{\mathbf{c}}_i \leftarrow \tilde{\mathbf{c}} \cdot \mathbf{G}_{B,Q}^{-1}([\frac{x^i}{x-2}])$
 6:     $c^i_{\mathsf{MPLWE}} \leftarrow \mathsf{REFHE.Repack}(n, Q, \mathsf{bspp.ksk}, \hat{\mathbf{c}})$
 7:     $c^i_{\mathsf{REFHE}} \leftarrow (x-2) \cdot c^i_{\mathsf{MPLWE}}$        ▷ $c^i_{\mathsf{REFHE}} = \text{REFHE}_Q(\mu_i x^i; \varepsilon''')$
 8:     $c' \leftarrow c' - \mathsf{ModSwitch}(c^i_{\mathsf{REFHE}}, q)$
 9:     $c_{\ll} \leftarrow c' \cdot x^{-i} \pmod q$
10: return $\sum_{i=0}^{n-1} c^i_{\mathsf{REFHE}}$
```

See supplementary material (Section D) for a discussion on correctness and security.

5.3 Boolean Operations

We now explain how to use our bootstrapping framework in order to perform boolean operations on the ciphertext in the course of bootstrapping. Our solution hinges on the property that the coefficients of the encoded plaintext μ are the bits of the message m, and those are the values that are recovered in the course of our bootstrapping procedure. We start by considering boolean operations that permute ciphertext bits, and move on to logical operations.

Permuting Ciphertext Bits. In Steps 5, 6, 7 of Algorithm 5.3 we construct $c^i_{\mathsf{REFHE}} = \mathrm{REFHE}_Q(\mu_i x^i; \varepsilon''')$. Let us consider a slight change to Step 5 and set $\hat{\mathbf{c}}_i \leftarrow \tilde{\mathbf{c}} \cdot \mathbf{G}^{-1}_{B,Q}([\frac{1}{x-2}])$, that is, the operand of $\mathbf{G}^{-1}_{B,Q}(\cdot)$ becomes independent of i. In this case we get $c^i_{\mathsf{REFHE}} = \mathrm{REFHE}_Q(\mu_i; \varepsilon''')$ instead of the above. However, we note that this is almost as good, since we can always multiply post-facto by x^j, for any j that we want, to obtain $c^{i,j}_{\mathsf{REFHE}} = \mathrm{REFHE}_Q(\mu_i x^j; \varepsilon''')$, at the cost of a factor $\hat{\gamma} \leq 3$ increase in the noise bound. In particular, $c^{i,i}_{\mathsf{REFHE}}$ would be the value c^i_{REFHE} from Step 7 in the algorithm, which allows to continue the execution of the bootstrapping loop as before.

After the end of the execution of the loop, we can compute immediately any $c^{i,j}_{\mathsf{REFHE}}$ that we want. Therefore, any shuffling of the bits may be implemented seamlessly. Let $\phi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1, \perp\}$ be any function, then we can return, in Step 10 of Algorithm 5.3, the value $\sum_{i=0}^{n-1} c^{\phi(i),i}_{\mathsf{REFHE}}$, where we syntactically define $c^{\perp,i}_{\mathsf{REFHE}}$ to indicate 0. By properly choosing ϕ, it is possible to implement circular/non-circular shifts (e.g. $\phi(i) = i + 1 \pmod n$ is a single-slot cyclic left shift), apply bit-masks, duplicate values, extract specific bits and perform similar operations. It is also possible to output more than one output ciphertext, e.g. store the lower $n/2$ bits in one ciphertext and the upper $n/2$ bits in another.

All of these operations incur a minimal penalty in terms of noise and runtime compered to ordinary bootstrapping.

Comparisons. An additional class of logical operations is producing a boolean value which corresponds to the truth value of some numerical comparison. It suffices to consider the setting where we have two input ciphertexts c_1, c_2 encrypting messages $\mathsf{m}_1, \mathsf{m}_2$, and we wish to recover c' that encrypts the value 1 if $\mathsf{m}_1 > \mathsf{m}_2$, and 0 otherwise. This is of course instrumental for branching operations, loop variables and such. This can be achieved straightforwardly using our techniques. First we can use arithmetic homomorphism to subtract $c_3 = c_2 - c_1$. Then we can bootstrap c_3 and extract a ciphertext containing only the most significant bit of the message encrypted by c_3. Indeed, by standard boolean logic (two's complement representation), the most significant bit is 1 if and only if the output is negative, which is the case if and only if $\mathsf{m}_1 > \mathsf{m}_2$.

Multi-Bit Boolean Operations. Let us now consider operations that are performed over multiple bits. We note that using the above, equality to 0 can be tested by checking that both $\mathsf{m} > -1$ and $\mathsf{m} < 1$, which can be done using two parallel bootstrapping sessions. Equality to 0 is also the n-bit NOR operation. Similarly other logical operators over the n bits can be implemented.

We may then turn our attention to performing bitwise AND, say between two numbers. That is, we have two plaintexts μ, μ', and we want to recover μ'' so that $\mu''_i = \mu_i \cdot \mu'_i$ for all i. This requires a bit more work, and we describe one possible method for performing this operation using a constant number of bootstrapping operations. First, we use a shuffling subroutine to split each input operand into two ciphertexts, where all even bits are set to 0, and the odd bits correspond to the odd bits of the original ciphertexts. That is, we have (from LSB to MSB): $\mu^{\text{lower}} = (\mu_0 0 \mu_1 0 \cdots \mu_{n/2} 0)$, $\mu^{\text{upper}} = (\mu_{n/2+1} 0 \cdots \mu_{n-1} 0)$, and likewise for μ'. We then add $c''^{\text{lower}} = c^{\text{lower}} + c'^{\text{lower}}$, $c''^{\text{upper}} = c^{\text{upper}} + c'^{\text{upper}}$, and notice that the *even* bits of the $c''^{\text{lower}}, c''^{\text{upper}}$ are exactly the bits of μ''. Another bootstrapping operation will allow to reorganize the bits and obtain a since c'' that encrypts μ'' as desired.

6 Performance

In this section we evaluate the performance of our scheme in comparison with BGV and TFHE, comparing run-time and ciphertext size. Our current implementation includes encryption, decryption, and all homomorphic arithmetic operations, namely, addition, multiplication, as well as modulus switching and key switching. Bootstrapping was not implemented. We note that the tests and results presented in this section were obtained with a straight-forward implementation in which no specific optimizations were applied to enhance performance. These results thus, reflect the raw computational costs associated with the building blocks of our scheme without the influence of advanced heuristics, algorithms or hardware-specific optimizations, that can be found in production-ready implementations for TFHE (tfhe-rs [za]) and BGV (HELib [HS20]).

Setup. Our experiments were conducted on an Apple M3 Pro laptop with 18GB RAM on a single thread. The benchmark code will be released as part of the library to ensure the reproducibility of our results.

Parameters. The parameters for all schemes are generated to reach a 128-bit security according to the lattice estimator [APS15]. We generated parameters for our scheme and for BGV so that correctness is achieved with high probability based on the theoretical analysis in Sections 4, C. We note that a heuristic based analysis as done in [HS20] should yield better parameters for both schemes, yet it is not expected to shift the advantage. We instantiate the key switching parameters with $d = 2$ and $q = q'$. Moreover, while the implementation supports working over module of general rank r, for the concrete parameters we chose $r = 1$. This achieves better efficiency for the key switching subroutine, and in addition provides a minimal ciphertext expansion factor.

Measurements. When referring to encryption, we measure the time it takes to encode, encrypt, and apply an initial modulus switch, on an input plaintext. When referring to homomorphic multiplication, we measure the step of tensor multiplication, key-switch, and modulo switch as a single multiplication component. We account for addition when considering zero multiplication depth, in both ciphertext size and "multiplication" runtime. The ciphertext size is measured after the first modulus switch, as this is the size required for transmission during communication, and this is also when we initiate the timing process for homomorphic multiplication.

Comparison with BGV. We generated parameters for the original BGV scheme [BGV12], ensuring that correctness is achieved with high probability, based on the analysis that involved bounding the ℓ_∞ norms of the noise, which were comparable to those obtained in our proposed scheme. This process was conducted for plaintext spaces of sizes 2^{16}, 2^{32}, and 2^{64}. Below, in Figure 1 we present a comparison of the ciphertext sizes between our scheme and BGV, using modulus sizes of 16-bit, 32-bit, and 64-bit, as a function of the supported arithmetic multiplication depth. Note that by zero multiplication depth, we refer to encryption that supports only a single addition. We see that the information rate is far better in our scheme, with factor ranging from 2 to over 7.

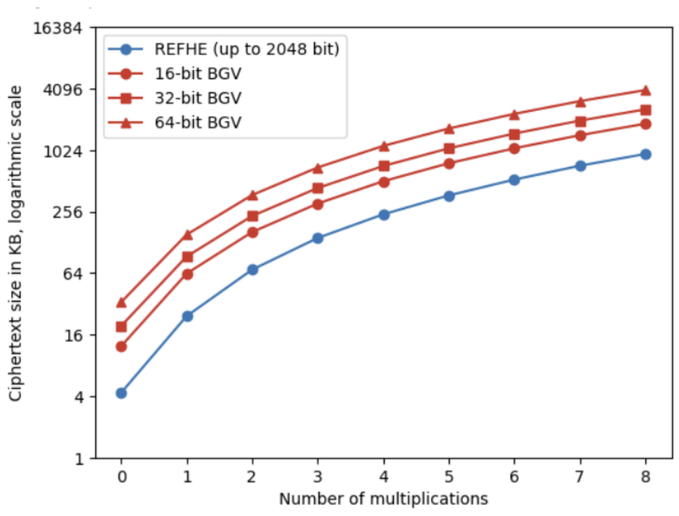


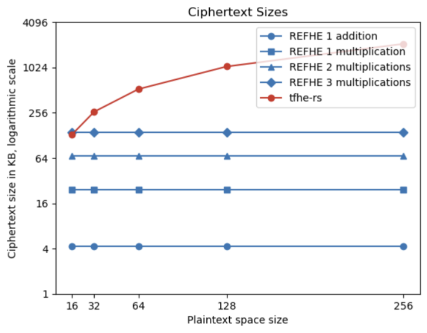
Fig. 1: Comparison with BGV of ciphertext size growth with homomorphic capacity for different plaintext spaces.

Note that the above parameters only support a leveled homomorphic scheme without bootstrapping. While we do not test our bootstrapping procedure experimentally, we provide our prediction as to its performance compared with HELib [HS21]. In HELib, the largest plaintext space of the form $\mathbb{Z}_p$ for which bootstrap run-time is reported is 8-bit [HS21, Table 4], which takes about 130 seconds to execute and 8.3GB space usage. We note that this exploits the technique of *thin-bootstrapping*, in which there is no packing and a single 8-bit plaintext is utilized per ciphertext. Without this technique, they report a 36 minute bootstrap of a batch of 21 8-bit plaintexts [HS21, Table 2], which improves the amortized time but yields a greater latency. In turn, they support 31 and 29 levels after bootstrap, respectively. In [GIKV23], run-time was improved by x2.6, but the circuit depth is greater. Moreover, their bootstrap technique involves digit extraction, and so the bootstrapping circuit depth increases with plaintext size. With our scheme, bootstrap is reduced to bootstrapping a TFHE ciphertext that encrypts a single bit, and our noise is only $\mathcal{O}(n)$ times larger due to the recursive nature of our bootstrap algorithm. In turn, we expect that ciphertext size will not significantly increase to support bootstrap.

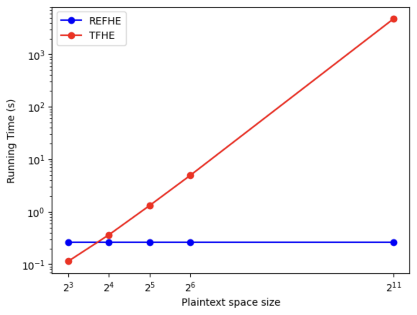
Comparison with tfhe-rs. The TFHE scheme requires a bootstrap operation for each computation. In Figure 2a, we present a comparison of ciphertext sizes between our proposed scheme and the tfhe-rs library, which supports operations on signed or unsigned integers ranging from 8-bit to 256-bit. The size of our ciphertext remains constant because, for a single addition, we operate over the ring $x^{1152} - x + 2$, which corresponds to arithmetic in $\mathbb{Z}/2^{1152}$. Consequently, arithmetic operations are performed in this large modulus, and the ciphertext size does not vary with the plaintext space.

In contrast, tfhe-rs achieves support for larger plaintext spaces by encrypting each byte individually. This results in a linear increase in ciphertext size with respect to the plaintext bit-length. However, this linear growth is less apparent in the graph due to the logarithmic scale employed.

The TFHE scheme requires a bootstrap along each operation. In Figure 2a we can see the comparison in the ciphertext size, between our scheme and the tfhe-rs library, that exposes operations on 8-bit to 256-bit signed or unsigned integers. Our graph is constant since in practice for a single addition we work over $x^{1152} - x + 2$ for a 128-bit security, which means that we perform arithmetic operations over $\mathbb{Z}/2^{1152}$. When choosing parameters for greater homomorphic capacity, the inital plaintext space is even larger. The way tfhe-rs work over large plaintext spaces is to essentially encrypt each byte individually, which causes a linear growth in the ciphertext space, which might not be clear from the graph, since the scale is logarithmic.



(a) Ciphertext vs plaintext size.
REFHE: 4kB, 24kB, 69kB, 140kB.
TFHE: 132kB, 263kB, 527kB, 1054kB, 2107kB.



(b) Multiplication Time.
REFHE (without bootstrapping): 0.2s.
TFHE: 0.1s, 0.35s, 1.3s, 4.9s.

Fig. 2: Comparison with TFHE.

We observe that the ciphertext size in REFHE is considerably smaller than in tfhe-rs, even when comparing against a 16-bit plaintext space and taking parameters that permit up to three multiplications prior to bootstrapping. This discrepancy becomes increasingly pronounced as the size of the plaintext space grows. Additionally, multiplication time in tfhe-rs grows quadratically with plaintext-space size, and takes more time than REFHE starting from 16-bit integers.

References

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Paper 2015/046, 2015.

BBPS19. Madalina Bolboceanu, Zvika Brakerski, Renen Perlman, and Devika Sharma. Order-LWE and the hardness of ring-LWE with entropic secrets. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II 25*, pages 91–120. Springer, 2019.

BC22. Iván Blanco-Chacón. On the rlwe/plwe equivalence for cyclotomic number fields. *Applicable Algebra in Engineering, Communication and Computing*, 33(1):53–71, 2022.

BCIV20. Carl Bootland, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Efficiently processing complex-valued data in homomorphic encryption. *J. Math. Cryptol.*, 14(1):55–65, 2020.

BCLvV16. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: reducing attack surface at low cost. Cryptology ePrint Archive, Paper 2016/461, 2016.

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

BK25. Dan Boneh and Jaehyung Kim. Homomorphic encryption for large integers from nested residue number systems. *IACR Cryptol. ePrint Arch.*, page 346, 2025.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

BTR24. Fabian Boemer, Karl Tarbe, and Rehan Rishi. Announcing swift homomorphic encryption, 2024. `https://www.swift.org/blog/announcing-swift-homomorphic-encryption/`.

BV11a. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011.

BV11b. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip

Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

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

CIV16. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Provably weak instances of ring-LWE revisited. Cryptology ePrint Archive, Paper 2016/239, 2016.

CJP21a. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander A. Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021.

CJP21b. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 1–19. Springer, 2021.

CKKS16. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

CLPX18. Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia. High-precision arithmetic in homomorphic encryption. In Nigel P. Smart, editor, *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, volume 10808 of *Lecture Notes in Computer Science*, pages 116–136. Springer, 2018.

Con09. Kieth Conrad. The different ideal. *Expository papers/Lecture notes. Available at: http://www. math. uconn. edu/ kconrad/blurbs/gradnumthy/different. pdf*, 2009.

DM14. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Paper 2014/816, 2014.

EHL14. Kirsten Eisentraeger, Sean Hallgren, and Kristin Lauter. Weak instances of PLWE. Cryptology ePrint Archive, Paper 2014/784, 2014.

ELOS15. Yara Elias, Kristin E. Lauter, Ekin Ozman, and Katherine E. Stange. Provably weak instances of ring-lwe, 2015.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

GC14. Matthias Geihs and Daniel Cabarcas. Efficient integer encoding for homomorphic encryption via ring isomorphisms. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2014.

GD84. Gary R. Greenfield and Daniel Drucker. On the discriminant of a trinomial. *Linear Algebra and its Applications*, 62:105–112, 1984.

Gen09a. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. *Symposium on the Theory of Computing*, page 169–178, 2009.

GH19. Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 438–464. Springer, 2019.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 465–482, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

GIKV23. Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren. On polynomial functions modulo pe and faster bootstrapping for homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–286. Springer, 2023.

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

GV24. Robin Geelen and Frederik Vercauteren. Fully homomorphic encryption for cyclotomic prime moduli. Cryptology ePrint Archive, Paper 2024/1587, 2024.

HS00. Jeffrey Hoffstein and Joseph H Silverman. Optimizations for ntru. In *Proc. the Conf. on Public Key Cryptography and Computational Number Theory, Warsaw*, pages 77–88, 2000.

HS20. Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

HS21. Shai Halevi and Victor Shoup. Bootstrapping for helib. *J. Cryptol.*, 34(1):7, 2021.

KDE$^+$24. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IEEE Trans. Computers*, 73(1):86–96, 2024.

LMW23. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 595–608. ACM, 2023.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

LPR13a. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

LPR13b. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 35–54. Springer, 2013.

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

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.

Pei16. Chris Peikert. How (not) to instantiate ring-LWE. Cryptology ePrint Archive, Paper 2016/351, 2016.

PP19. Chris Peikert and Zachary Pepin. Algebraically structured LWE, revisited. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17*, pages 1–23. Springer, 2019.

PP24. Chris Peikert and Zachary Pepin. Algebraically structured lwe, revisited. *J. Cryptol.*, 37(3):28, 2024.

PRS17. Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-lwe for any ring and modulus. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 461–473. ACM, 2017.

RAD$^+$78. Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

RSSS17. Miruna Rosca, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Middle-product learning with errors. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2017.

RSW18. Miruna Rosca, Damien Stehlé, and Alexandre Wallet. On the ring-LWE and polynomial- LWE problems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 146–173. Springer, 2018.

SEA11. Microsoft SEAL. Microsoft seal is an easy-to-use and powerful homomorphic encryption library., 2011. `https://github.com/Microsoft/SEAL`.

SS10. Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 617–635. Springer, 2009.

tfh17. tfhe. Tfhe: Fast fully homomorphic encryption library over the torus, 2017. https://github.com/tfhe/tfhe.

za. zama ai. Tfhe-rs: A pure rust implementation of the tfhe scheme for boolean and integer arithmetics over encrypted data. https://github.com/zama-ai/tfhe-rs.

Zam21. Zama. Concrete: Tfhe compiler that converts python programs into fhe equivalent, 2021. https://github.com/zama-ai/concrete.

A Standard Definitions from Cryptographic Literature

A.1 Homomorphic Encryption

We now define homomorphic encryption and its desired properties. Throughout this section (and this work) we use κ to indicate the security parameter.

A homomorphic (public-key) encryption scheme

$$\mathrm{HE} = (\mathrm{HE.Keygen}, \mathrm{HE.Enc}, \mathrm{HE.Dec}, \mathrm{HE.Eval})$$

is a quadruple of PPT algorithms as follows:

- **Key generation:** The algorithm $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathrm{HE.Keygen}(1^\kappa, \mathrm{aux})$ takes a unary representation of the security parameter and auxilary input that in our case encodes the plaintext space and the homomorphic capcity and outputs a public encryption key pk, a public evaluation key evk, and a secret decryption key sk.
- **Encryption:** The algorithm $c \leftarrow \mathrm{HE.Enc}_{\mathsf{pk}}(\mu)$ takes the public key pk and a message $\mu \in \mathbb{Z}/p\mathbb{Z}$ and outputs a ciphertext c.
- **Decryption:** The algorithm $\mu^* \leftarrow \mathrm{HE.Dec}_{\mathsf{sk}}(c)$ takes the secret key sk and a ciphertext c and outputs a message $\mu^* \in \mathbb{Z}/p\mathbb{Z}$.
- **Homomorphic evaluation:** The algorithm $c_f \leftarrow \mathrm{HE.Eval}_{\mathsf{evk}}(f, c_1, \ldots, c_\ell)$ takes the evaluation key evk, a function $f : (\mathbb{Z}/p\mathbb{Z})^\ell \to \mathbb{Z}/p\mathbb{Z}$, and a set of ℓ ciphertexts $c_1, \ldots, c_\ell$, and outputs a ciphertext c_f.

Definition A.1. *A homomorphic encryption scheme is said to correctly evaluate a circuit family F if for all $f \in F$ and for all $m_1, \ldots, m_\ell \in \mathbb{Z}/p\mathbb{Z}$, the following holds: If sk, pk are correctly generated by* HE.Keygen *with security parameter κ, and if $c_i = \mathrm{HE.Enc}_{\mathsf{pk}}(m_i)$ for all i, and $c_f = \mathrm{HE.Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_\ell)$, then*

$$\Pr[\text{HE.Dec}_{\mathsf{sk}}(c_f) \neq f(m_1, \ldots, m_\ell)] = \text{negl}(\kappa),$$

where the probability is taken over all randomness in the error and key distributions.

Furthermore, the scheme is said to compactly evaluate the family if, in addition, the running time of the decryption circuit depends only on κ and not on its input.

The only security notion we consider in this chapter is semantic security, namely security with respect to passive adversaries. We use its widely known formulation as IND-CPA security, defined as follows.

Definition A.2 (IND-CPA Security).
Let $(KeyGen, Enc, Dec, Eval)$ be a public key encryption scheme. We define an experiment $Expr_b^{cpa}[\mathcal{A}]$ parameterized by a bit $b \in \{0,1\}$ and an efficient adversary $\mathcal{A}$:

$$Expr_b^{cpa}[\mathcal{A}](1^\kappa) : \begin{cases} (sk, pk, ek) \leftarrow KeyGen(1^\kappa) \\ (x_0, x_1) \leftarrow \mathcal{A}(1^\kappa, pk, ek) \\ ct \leftarrow Enc_{pk}(x_b) \\ b' \leftarrow \mathcal{A}(ct) \\ Return\ b' \end{cases}$$

We say that the scheme is IND-CPA secure if for every PPT adversary A, it holds that

$$Adv^{cpa}[A](\kappa) := |\Pr\{Expr_0^{cpa}[A](1^\kappa) = 1\} - \Pr\{Expr_1^{cpa}[A](1^\kappa) = 1\}| = negl(\kappa).$$

A.2 Gadget Decomposition

We present a ring generalization of the widely used "gadget decomposition" technique [MP12]. For a ring $\mathcal{R}$, a *gadget vector* $\mathbf{g} \in \mathcal{R}^l$, is a row vector of ring elements, equipped with a (not necessarily linear) *decomposition operator* $\mathbf{g}^{-1} : \mathcal{R} \to \mathcal{R}^l$. We require that for all $x \in \mathcal{R}$, it holds that $\mathbf{g} \cdot \mathbf{g}^{-1}(x) = x$. Namely the gadget is an "inverse" of the decomposition operator. Importantly, the gadget is defined so as to ensure that for all $x \in \mathcal{R}$ it holds that $\mathbf{g}^{-1}(x)$ is "small", with respect to some measure. Thus the decomposition operator allows to trade-off size for dimension, while allowing linear reconstruction.

The *gadget matrix* $\mathbf{G}_\mathbf{g}$ corresponding to a gadget vector $\mathbf{g}$ is defined as $\mathbf{G}_\mathbf{g} = \mathbf{I}_k \otimes \mathbf{g} = \text{diag}(\mathbf{g}, \ldots, \mathbf{g}) \in \mathcal{R}^{k \times kl}$ (where the diag operators organizes its operands along the block-diagonal of a matrix, with 0 in the off-diagonal), where $\mathbf{I}_k$ is the identity matrix of size k. The decomposition operation, $\mathbf{G}_\mathbf{g}^{-1} : \mathcal{R}^{k \times k'} \to \mathcal{R}^{kl \times k'}$, applies $\mathbf{g}^{-1}$ coordinate-wise, expanding each entry into an l-dimensional column vectors.

More generally we may consider a gadget matrix that is generated by a set of different gadget vectors: $\mathbf{G}_{\mathbf{g}_1,\ldots,\mathbf{g}_k}$. This is defined as $\mathbf{G}_{\mathbf{g}_1,\ldots,\mathbf{g}_k} = \text{diag}(\mathbf{g}_1, \ldots, \mathbf{g}_k) \in$

$\mathcal{R}^{k\times(\sum l_i)}$. The decomposition operation, denoted as $\mathbf{G}^{-1}_{\mathbf{g}_1,\ldots,\mathbf{g}_k} : \mathcal{R}^{k\times k'} \to \mathcal{R}^{(\sum l_i)\times k'}$ is defined analogously.

Next we define a specific class of gadgets that are commonly used in the paper.

Definition A.3 (The Powers Gadget). *Let $\mathcal{R}$ be either a polynomial ring $\mathbb{Z}[x]/\langle f(x)\rangle$ or $\mathbb{Z}$ itself. For $x \in \mathcal{R}_q$ we define* $\text{Decomp}_d(x)$ *be the decomposition of x into the following base d representation, if $x = \sum_{j=0}^{\lceil \log_d(q)\rceil}(b_j d^j)$ for $b_j \in \mathcal{R}/d$ with coefficients smaller than $\lfloor d/2\rfloor$,* $\text{Decomp}_d(x) = (b_0,\ldots,b_{\lceil \log_d(q)\rceil}) \in \mathcal{R}/d$. *The* $\text{Decomp}_d(x)$ *operator is the decoposition operator $\mathbf{g}^{-1}_{\text{Powers}_d}$ corresponding to the* Powers_d *gadget vector:* $\text{Powers}^q_d = (1, d, \ldots, d^{\lceil \log_d(q)\rceil}) \in \mathcal{R}_q^{\lceil \log_d(q)\rceil}$. *We Notice that $\ell_\infty(\mathbf{g}^{-1}_{\text{Powers}_d}(x)) \le \lfloor d/2\rfloor$.*

B Correctness and Security of the Basic Scheme

To keep track of the noise, we have the following definition:

Definition B.1. *Let $\mathbf{c} \in \mathcal{R}^{r+1}_{\mathfrak{q}}$ be a ciphertext, let $\mathbf{s} \in \mathcal{R}^{r+1}$, and let $\mu \in \mathcal{R}_{\mathfrak{p}}$. We define*

$$\eta_{\mathbf{s}}(\mathbf{c},\mu) = \min\left\{\ell_\infty(\mu+e) : e \in \mathfrak{p},\ \mu + e = \mathbf{s}\cdot\mathbf{c} \pmod{\mathfrak{q}}\right\} .$$

We omit the subscript when $\mathbf{s}$ is clear from the context. We also omit μ when clear from context

Lemma B.1. *If $\eta_{\mathbf{s}}(\mathbf{c},\mu) \le \lfloor (q-1)/2\rfloor$, then* $\text{REFHE.Dec}_{\mathbf{s}}(\mathbf{c}) = \mu$.

Proof. By the assumption there is $e \in \mathfrak{p}$ such that $\mathbf{s}\cdot\mathbf{c} = \mu + e \mod q$. Since $\ell_\infty(\mu+e) \le \lfloor (q-1)/2\rfloor$ we also have:

$$[\mathbf{s}\cdot\mathbf{c}]_{\mathcal{R}_q} = \mu + e$$

Now $(\mu+e) \mod \mathfrak{p} = \mu$ as we wanted.

In the following sections we will use Lemma B.1 when analyzing correctness of homomorphic evaluation. Indeed, given a function $f : (\mathbb{Z}/p)^N \to \mathbb{Z}/p$ and input ciphertexts $\mathbf{c}_1 = \text{REFHE.Enc}(\mu_1), \ldots, \mathbf{c}_N = \text{REFHE.Enc}(\mu_N)$, let $\mathbf{c}$ be the output ciphertext of the homomorphic evaluation. Then by Lemma B.1 it suffices to bound $\eta(\mathbf{c},\mu)$ for $\mu = \mathsf{Encode}\circ f \circ \mathsf{Decode}(\mu_1,\ldots,\mu_N)$.

First, we analyse correctness of REFHE as an encryption scheme, without applying additional homomorphic operations:

Lemma B.2. *Let* $\mathbf{c} = \text{REFHE.Enc}_{\mathsf{pk}}(\mu)$. *Denote by $q_0/2 := (1 + 2\cdot\hat{\gamma}\cdot\ell_1(\mathbf{s}))\cdot B_{\chi_e} + 1$. Then $\eta_{\mathbf{s}}(\mathbf{c},\mu) \le q_0/2$.*

Proof. Notice that

$$\begin{aligned}\mathbf{s}\cdot\mathbf{c} &= \mathbf{b}\cdot\mathbf{r}+e_0+\mu-\mathbf{s}_1\cdot(\mathbf{A}\mathbf{r}+\mathbf{e}_1)\\ &= (\mathbf{s}_1\mathbf{A}+\mathbf{e})\mathbf{r}+e_0+\mu-\mathbf{s}_1\mathbf{A}\mathbf{r}+\mathbf{s}_1\cdot\mathbf{e}_1\\ &= \mu+\mathbf{r}\cdot\mathbf{e}+e_0+\mathbf{s}_1\cdot\mathbf{e}_1\end{aligned}$$

Where $\mathbf{s}_1,\mathbf{r}\leftarrow\chi_s$, $\mathbf{e},\mathbf{e}_1\in\chi_e^r$, $e_0\in\chi_e$, and $\ell_\infty(\mu)=1$. It follows that $\mathbf{c}\cdot\mathbf{s}=\mu+e_3$ when $e_3\leq(1+2\cdot\hat{\gamma}\cdot\ell_1(\mathbf{s}))\cdot B_{\chi_e}+1$

Lemma B.3. *The public-key encryption scheme* REFHE *is IND-CPA secure based on the* MPLWE *assumption (Definition 3.2) with parameters* $\mathsf{mplweparams}=(f,n,r,\mathfrak{q},\chi_s,\chi_e)$.

Proof (Proof Sketch.). The proof of this lemma follows the standard argument for proving security for encryption schemes based on LWE-style assumptions. See, e.g., [Reg05, LPR10]. The proof follows by a hybrid argument where we consider an adversary that is given the public key pk and a ciphertext $\mathbf{c}$ which are properly generated by the scheme and encrypt some value μ. We recall Remark 3.2 with $t=(x-2)$.

We first replace the $\mathbf{b}$ vector in the public key to be sampled uniformly rather than being computed using MPLWE. By the MPLWE assumption with secret $\mathbf{s}_1$, this hybrid is computationally indistinguishable from the original experiment.

Then we replace $\mathbf{c}$ by a uniform vector. This is computationally indistinguishable from the previous hybrid again relying on MPLWE with secret $\mathbf{r}$.

Finally, $\mathbf{c}$ is completely uniform and independent of μ, which implies the security of the scheme.

In Section E, we relate the hardness of MPLWE to that of more commonly used assumptions in lattice-based cryptography, such as RLWE, and discuss the proper choice of parameters.

C Homomorphic Arithmetic Operations

In this section we present our algorithms for the homomorphic evaluation of arithmetic operations: addition, multiplication by a scalar and multiplication of ciphertexts. For the sake of the multiplication operations, we also introdce our versions of the key switching and modulus switching techniques.

Homomorphic evaluation of logical operations will be implemented as a part of our bootstrapping process, see Section 5.3 for details.

During this section there are addition and multiplication between elements that naturally live in different spaces. some of them are in $\mathcal{R}_q$ (the ciphertexts), some in $\mathfrak{p}$ (the errors), some in $\mathcal{R}/\mathfrak{p}$ (the messages), and some in $\mathcal{R}$ (the secret keys). Although they live in different spaces, the operations and equalities in this section are in $\mathcal{R}_q$, unless stated otherwise.

Our convention in this section is that $e_j\in\mathfrak{p}$, μ_j is the message.

C.1 Addition

REFHE.Add takes two ciphertexts encrypted under the same secret key $\mathbf{s}$. The addition is performed by adding the two ciphertexts (as vectors of ring elements).

Algorithm C.1: REFHE.Add **(Homomorphic Addition)**

$\mathbf{c}_3 \leftarrow \text{REFHE.Add}(\mathbf{c}_1, \mathbf{c}_2)$: Output $\mathbf{c}_3 \leftarrow \mathbf{c}_1 + \mathbf{c}_2$

Lemma C.1. *For* $\mathbf{c}_3 = \text{REFHE.Add}(\mathbf{c}_1, \mathbf{c}_2)$ *we have* $\eta(\mathbf{c}_3, \mu_1+\mu_2) \leq \eta(\mathbf{c}_1, \mu_1)+\eta(\mathbf{c}_2, \mu_2)$.

Proof. Let $\mathbf{s} \cdot \mathbf{c}_1 = \mu_1 + e_1$, $e_1 \in \mathfrak{p}$ such that $\ell_\infty(\mu_1 + e_1) = \eta_{\mathbf{s}}(\mu_1 + e_1)$, $\mathbf{s} \cdot \mathbf{c}_2 = \mu_2 + e_2$, $e_2 \in \mathfrak{p}$ such that $\ell_\infty(\mu_2 + e_2) = \eta_{\mathbf{s}}(\mu_2 + e_2)$. Then $\mathbf{s} \cdot (\mathbf{c}_1 + \mathbf{c}_2) = \mu_1 + e_1 + \mu_2 + e_2 = (\mu_1 + \mu_2) + (e_1 + e_2)$.

C.2 Scalar Multiplication

Same as with the message, we have to encode the scalar as a polynomial that equal to the scalar mod $\mathfrak{p}$. Then multiply each coordinate of the ciphertext by it. Notice that since the norm of the encoded scalar is small, when multiplying by it the noise grows roughly by the expansions factor, which is logarithmic in the size of the plaintext space. In BGV this factor is linear in the size of the plaintext space, so for large plaintext spaces we get a significant improvement in the noise growth.

Algorithm C.2: Multiplication by a scalar

$\mathbf{c}' \leftarrow \text{REFHE.ScalarMult}(\alpha, \mathbf{c})$: Outputs $\mathbf{c}' = \text{REFHE.}\mathsf{Encode}(\alpha) \cdot \mathbf{c}$.

Lemma C.2. *Let* $\mathbf{c}$ *be a ciphertext, then for* $\mathbf{c}' = \text{REFHE.ScalarMult}(\mathbf{c}, \alpha)$ *we have* $\eta_{\mathbf{s}}(\mathbf{c}', \alpha \cdot \mu) \leq \gamma \cdot B_{\mathsf{enc}} \cdot \eta_{\mathbf{s}}(\mathbf{c}, \mu)$.

Proof. Let $\mathbf{s} \cdot \mathbf{c}' = \mu + e$ when $\ell_\infty(\mu + e) = \eta_{\mathbf{s}}(\mu + e)$. Observe that

$$\mathbf{s} \cdot \mathbf{c}' = \text{REFHE.}\mathsf{Encode}(\alpha) \cdot \mu + \text{REFHE.}\mathsf{Encode}(\alpha) \cdot e$$

Since $\ell_\infty(\text{REFHE.}\mathsf{Encode}(\alpha)) \leq B_{\mathsf{enc}}$, we have

$$\ell_\infty(\text{REFHE.}\mathsf{Encode}(\alpha) \cdot \mu + \text{REFHE.}\mathsf{Encode}(\alpha) \cdot e) \leq \gamma \cdot B_{\mathsf{enc}} \cdot \ell_\infty(\mu + e)$$

Also $\text{REFHE.}\mathsf{Encode}(\alpha) \cdot \mu + \text{REFHE.}\mathsf{Encode}(\alpha) \cdot e \mod \mathfrak{p} = \alpha \cdot \text{REFHE.}\mathsf{Decode}(\mu) \mod \mathfrak{p}$ as we wanted

C.3 Key switching

The following Key switching technique is based on the one presented in [HS20], and is done in order to reduce the relative noise, in comparison to the standard approach in [BGV12]. This is indeed a generalization of the BGV keyswitching, which can be obtained as a special case if $q = q'$.

Algorithm C.3: Keyswitch

1. **Key Generation**: $\mathrm{SwitchKeyGen}_{\mathbf{G}}(q, q', \mathbf{s}_1 \in \mathcal{R}_q^{r_1}, \mathbf{s}_2 \in \mathcal{R}_{q'}^{r_2})$ for $q|q'$ outputs a matrix $\tau_{\mathbf{s}_1 \to \mathbf{s}_2}$ over $\mathcal{R}_{q'}$, satisfying:
$$\mathbf{s}_2 \cdot \tau_{\mathbf{s}_1 \to \mathbf{s}_2} = T \cdot \mathbf{s}_1 \cdot \mathbf{G} + \mathbf{e} \pmod{q'}$$
2. **Key Switch**: $\mathrm{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1, w)$ outputs $\mathbf{c}_2 = \tau_{\mathbf{s}_1 \to \mathbf{s}_2} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') \in (\mathcal{R}_{q'})^{r_2}$ for $T = q'/q$, and $\mathbf{c}_1' = w \cdot \mathbf{c}_1 \mod q$.

w will be chosen as an elemnt in $\mathcal{R}_{\{0,1\}}$ In order for everything to be defined $T, \mathfrak{p}$ need to be co-prime. In practice, we always take $q'|q$ odd integers, so this condition is satisfied.

Lemma C.3. *Let* $\mathbf{s}_1, \mathbf{s}_2, q, q'$ *be as in* $\mathrm{SwitchKeyGen}(q, q', \mathbf{s}_1, \mathbf{s}_2)$. *Let* $\mathbf{c}_1 \in \mathcal{R}_q^{r_1}$, $\mathbf{c}_1' = w \cdot \mathbf{c}_1 \mod q$ *and* $\mathbf{c}_2 \leftarrow \mathrm{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1, w)$. *Then,*

$$\mathbf{s}_2\mathbf{c}_2 = \mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') + T \cdot \mathbf{s}_1 \cdot \mathbf{c}_1' \mod q'$$

Proof. Let $\mathbf{A} = \tau_{\mathbf{s}_1 \to \mathbf{s}_2}$. We have:

$$\begin{aligned}
\mathbf{s}_2 \cdot \mathbf{c}_2 &= \mathbf{s}_2 \cdot \mathbf{A} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') \\
&= (T \cdot \mathbf{s}_1 \cdot \mathbf{G} + \mathbf{e}) \cdot \mathbf{G}^{-1}(\mathbf{c}_1') \\
&= \mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') + T \cdot (\mathbf{s}_1 \cdot \mathbf{c}_1' + qE) \\
&= \mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') + T \cdot (\mathbf{s}_1 \cdot \mathbf{c}_1' + qE)
\end{aligned}$$

When the equations are mod q'.

$$\mathbf{s}_2 \cdot \mathbf{c}_2 = \mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') + T \cdot \mathbf{s}_1 \cdot \mathbf{c}_1' \mod q'$$

Corollary C.1. *Let* $\mathbf{c}_1 \in \mathcal{R}_q^{r_1}$ *and* $\mathbf{c}_2 \leftarrow \mathrm{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1)$. *Then for the powers of d gadget* $\mathbf{G}_{\mathrm{Powers}_d}$ *(defined in Definition A.3) we have that* $\tau_{\mathbf{s}_1 \to \mathbf{s}_2}$ *is in* $\mathcal{R}_{q'}^{r_1 \cdot \lceil \log_d(q) \rceil \times r_2}$*, and:*

$$\eta_{\mathbf{s}_2}(\mathbf{c}_2, \mu w \cdot T \mod \mathfrak{p}) \leq q'/q \cdot \gamma_w \cdot \eta_{\mathbf{s}_1}(\mathbf{c}_1, \mu) + r_1 \cdot \lfloor d/2 \rfloor \cdot \lceil \log_d(q) \rceil (B_{\chi_e}(q')) \cdot \gamma$$

Proof. Notice that

$$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{c}_1') + T \cdot \mathbf{s}_1 \cdot \mathbf{c}_1' \mod \mathfrak{p} = (Tw \cdot \mu + Tw \cdot e + e') \mod \mathfrak{p} = Tw\mu$$

Lemma C.4 (Security). *For every known* $\mathbf{s}_1 \in \mathcal{R}_q^{r_1}$ *and a random* $\mathbf{s}_2 \leftarrow \chi_s$*, the* $\mathrm{SwitchKeyGen}(s_1, s_2)$ *distribution is computationally indistinguishable from uniform.*

Relinearization. Typically, homomorphic multiplication outputs a ciphertext encrypted under $\mathbf{s}\otimes\mathbf{s}$. In order to keep the dimension fixed, it is common to wrap this procedure with a *relinearization* procedure that switches the key from $\mathbf{s}\otimes\mathbf{s}$ back to $\mathbf{s}$. The relinearization key is $\mathrm{SwitchKeyGen}(\mathbf{s}, \mathbf{s}\otimes\mathbf{s})$, which is considered as part of the public key. We can then invoke REFHE.SwitchKey from $\mathbf{s}\otimes\mathbf{s}$ to $\mathbf{s}$ after every multiplication.

C.4 Modulus Switching

We denote $y = \lceil a \rfloor^{c+\mathfrak{p}}$ the 1-rounding algorithm with respect to ℓ_∞ for the lattice $\mathfrak{p}$, meaning for $a \in \mathbb{Q}[x]/f(x)$, $c \in \mathcal{R}$. finding a $y \in \mathcal{R}$ such that $(y - c) \in p$, $\ell_\infty(y - a) \leq 1$.

Algorithm C.4: Rounding

Given an input a, output $\lceil a \rfloor_{c+\mathfrak{p}} = \lfloor a \rfloor + \mathsf{Encode}(c - \lfloor a \rfloor)$ where $\lfloor\cdot\rfloor$ is performed saperetly to each coefficient in the coefficient embedding.

Lemma C.5. *Algorithm C.4 is a 1-rounding algorithm to the lattice* $\mathfrak{p}$ *with respect to* ℓ_∞*. Meaning* $\ell_\infty(\lceil a \rfloor_{c+\mathfrak{p}}) \leq 1$, $\lceil a \rfloor_{c+\mathfrak{p}} = c \mod \mathfrak{p}$

Proof. Let $a' = \lfloor a \rfloor$, $b = c(2) - a'(2) \mod (2^n)$, $c' = \lceil a \rfloor_{c+\mathfrak{p}}$. We need to prove that $c' - c \in \mathfrak{p}$ and that $\ell_\infty(a - c') \leq 1$. For the second claim we have

$$|a^{(i)} - c'^{(i)}| = |b_i - \{a^{(i)}\}| \leq 1$$

And regarding the first claim, we have:

$$a'(2) + \sum b_i 2^i = c(2) \mod 2^n$$

Now using the fact that x divides $x - x^{n-1} = 2$ we get:

$$\begin{aligned} a'(2) + \sum b_i 2^i &= c(2) \mod x^n \\ a'(2) + \sum b_i 2^i &= c(2) \mod (x-2) \\ c'(x) = a'(x) + \sum b_i x^i &= c(x) \mod (x-2) \end{aligned}$$

As desired.

Algorithm C.5: Modulus Switching

For $\mathbf{c} \in \mathcal{R}_q^{r+1}$, $\mathbf{c}' \leftarrow \mathsf{ModSwitch}_{q,q',w}(\mathbf{c})$, outputs $\mathbf{c}' \in \mathcal{R}_{q'}^{r+1}$: $\mathbf{c}'[i] = \lceil \frac{wq'}{q} \cdot \mathbf{c}[i] \rfloor_{q^{-1}q'w\mathbf{c}[i]+\mathfrak{p}}$, for $0 < i \leq r+1$, for $w \in \mathcal{R}_{\{0,1\}}$

Notice that since the ciphertext contains multiple elements from $\mathcal{R}$, the rounding is done on each element seperately.

We start with a lemma on the error rate of modulus switching for (rational) integer moduli. While this is the most efficient modulus switching we have for our scheme, it provides a good sense of the underlying concepts.

Lemma C.6. *Assume that $q, q' \in \mathbb{Z}$ and let $\mathbf{c}$ be a ciphertext. Then for $\mathbf{c}' = \mathsf{ModSwitch}_{q,q',w}(\mathbf{c})$,*

$$\eta(\mathbf{c}', \mu \cdot \frac{wq'}{q}) \leq \gamma_w \cdot \frac{q'}{q}\eta(\mathbf{c}, \mu) + \hat{\gamma}\ell_1(\mathbf{s})$$

Proof. We have,

$$\mathbf{s} \cdot \mathbf{c} = \mu + \epsilon + qE$$

and

$$\frac{q'w}{q}\mathbf{s} \cdot \mathbf{c} = \frac{q'w}{q}\mu + \frac{q'w}{q}\epsilon + q'wE.$$

We have

$$\mathbf{s} \cdot \mathbf{c}' = \frac{q'w}{q}\mu + \frac{q'w}{q}\epsilon + q'wE + \mathbf{s} \cdot (\mathbf{c}' - \frac{q'w}{q}\mathbf{c}).$$

Denote $v = \frac{q'w}{q}\mu + \frac{q'w}{q}\epsilon + \mathbf{s} \cdot (\mathbf{c}' - \frac{q'w}{q}\mathbf{c})$. then:

$$\begin{aligned}\ell_\infty(v) &\leq \frac{q'}{q}\gamma_w \cdot \eta(\mathbf{c}, \mu) + \ell_\infty(\mathbf{s} \cdot (\mathbf{c}' - \frac{q'w}{q}\mathbf{c})) \\ &\leq \frac{q'}{q}\gamma_w \cdot \eta(\mathbf{c}, \mu) + \hat{\gamma} \cdot \ell_1(\mathbf{s})\end{aligned}$$

we also have

$$\mathbf{s} \cdot \mathbf{c}' - q'wE = v.$$

We have that $\mathbf{s} \cdot \mathbf{c}' \mod \mathfrak{p} = \frac{q'w}{q}\mathbf{s} \cdot \mathbf{c} \mod \mathfrak{p}$. Therefore,

$$(\mathbf{s} \cdot \mathbf{c}' - q'wE) \mod \mathfrak{p} = \frac{q'w}{q}(\mathbf{s} \cdot \mathbf{c} - qE) \mod \mathfrak{p},$$

which gives us

$$v \mod \mathfrak{p} = \frac{q'w}{q}\mu$$

So to conclude, $\mathbf{s} \cdot \mathbf{c}' \mod q' = v$, for v s.t. $\ell_\infty(v) \leq \frac{q'w}{q} \cdot \eta(\mathbf{c}, \mu) + \hat{\gamma} \cdot \ell_1(\mathbf{s})$, and $v \mod \mathfrak{p} = \frac{q'w}{q}\mu$. This completes the proof.

Remark C.1. If $q = q' \mod \mathfrak{p}$ and multiplication by w is trivial and we can see from the proof that $\eta(\mathbf{c}') \leq \frac{q'}{q}\eta(\mathbf{c}) + \hat{\gamma}\ell_1(s)$ which removes the γ_w from the noise bound. As mentioned at the beginning of this section, it may not be ideal to choose integers q and q' that are congruent modulo $\mathfrak{p}$, as this could lead to very large ciphertext moduli values. Specifically, this condition implies $q \equiv q' \pmod{p}$, resulting in $q \geq p$. This would require a much larger ciphertext modulus than what is typically used in our implementations. For that we will have to choose $w \neq 1$ and suffer from a multiplicative expansion factor in the resulting error. In order to overcome that we introduce the ideal modulus switching, which will result in working modulo $\mathfrak{q}$ which is not a rational number.

C.5 Multiplication

Algorithm C.6: Tensor multiplication

$\mathbf{c}_3 \leftarrow$ REFHE.TensorMult$(\mathbf{c}_1, \mathbf{c}_2)$ gets $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{R}_q^{r+1}$ and returns $\mathbf{c}_3 = \mathbf{c}_1 \otimes \mathbf{c}_2 \in \mathcal{R}_q^{(r+1)^2}$

Given ciphertexts encrypting μ_1, μ_2 under $\mathbf{s}$, the algorithm returns ciphertext that encrypts $\mu_1 \cdot \mu_2 \mod \mathfrak{p}$ under $\mathbf{s} \otimes \mathbf{s}$. This is formalized via the following Lemma:

Lemma C.7. *Let $\mathbf{c}_1, \mathbf{c}_2$ be ciphertexts that encrypt μ_1, μ_2 under $\mathbf{s}_1, \mathbf{s}_2$ respectively. Then*

$$\eta_{\mathbf{s}\otimes\mathbf{s}}(\text{REFHE.TensorMult}(\mathbf{c}_1, \mathbf{c}_2), \mu_1 \cdot \mu_2) \leq \gamma \cdot \eta_{\mathbf{s}}(\mathbf{c}_1, \mu_1) \cdot \eta_{\mathbf{s}}(\mathbf{c}_2, \mu_2)$$

Proof. Let $\mathbf{s} \cdot \mathbf{c}_1 = e_1$, $\mathbf{s} \cdot \mathbf{c}_2 = e_2$, where $e_1 = \mu_1 + e_1'$, $e_2 = \mu_2 + e_2'$, $e_1', e_2' \in \mathfrak{p}$ and $\eta_{\mathbf{s}}(\mathbf{c}_1, \mu_1) = \ell_\infty(e_1)$, $\eta_{\mathbf{s}}(\mathbf{c}_2, \mu_2) = \ell_\infty(e_2)$. Observe that

$$(\mathbf{s} \otimes \mathbf{s}) \cdot (\mathbf{c}_1 \otimes \mathbf{c}_2) = (\mathbf{s} \cdot \mathbf{c}_1) \cdot (\mathbf{s} \cdot \mathbf{c}_2) = e_1 \cdot e_2$$

and $e_1 \cdot e_2 \mod \mathfrak{p} = (\mu_1 + e_1') \cdot (\mu_2 + e_2') \mod \mathfrak{p} = \mu_1 \cdot \mu_2$. The lemma follows.

The dimension of REFHE.TensorMult$(\mathbf{c}_1, \mathbf{c}_2)$ is squared compared to the input ciphertexts. When multiplying, we therefore first compute $\mathbf{c}_4 =$ REFHE.SwitchKey$(\tau_{\mathbf{s}\otimes\mathbf{s}\to\mathbf{s}}, \mathbf{c}_3)$ and then $\mathbf{c}_5 = \mathsf{ModSwitch}_{q,q',w}(\mathbf{c}_4)$ for q, q' determined in the scheme parameters, in order to reduce the noise.

Algorithm C.7: Multiplication

$\mathbf{c}_3 \leftarrow$ REFHE.Mult$(\mathbf{c}_1, \mathbf{c}_2)$ gets $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{R}_q^{r+1}$, sets

$$\mathbf{c}_4 = \text{REFHE.SwitchKey}(\tau_{\mathbf{s}\otimes\mathbf{s}\to\mathbf{s}}, \text{REFHE.TensorMult}(\mathbf{c}_1, \mathbf{c}_2), w)$$

and returns

$$\mathbf{c}_3 = \mathsf{ModSwitch}_{q,q',w'}(\mathbf{c}_4)$$

Notice that w, w' can be chosen in $\mathcal{R}_{\{0,1\}}$ such that $\mathbf{c}_3$ encrypts the multiplication of the messages $\mathbf{c}_1, \mathbf{c}_2$ encrypt (given that the noise is small), and not a scalar multiplication of it.

Moduli Ladder. As in [BGV12], the setup phase of the scheme in algorithm 4.1 produces also a "ladder" of the cipher text moduli that will be used during the evaluation of the scheme. Notice that after each multiplication we perform key switch and then modulus switch, so the evaluation key should consist of a Public keyswitch matrix for each multiplication before the bootstrapping, and the relevant modulus.

C.6 Optimizations

Optimization for keyswitch and modulus switch. Notice that there is a multiplication by a scalar in both algorithms: multiplying $\mathbf{c}_1$ by w in Algorithm C.3, and multiplying $\mathbf{c}$ by w in Algorithm C.5. It turns out that we can ignore these scalar multiplications and account for them only in the final step. Specifically, by skipping these scalar multiplications, i.e., setting $w = w' = 1$, and given bounds on $\eta_{\mathbf{s}}(\mathbf{c}_1, \mu_1)$ and $\eta_{\mathbf{s}}(\mathbf{c}_2, \mu_2)$, Algorithm C.7 provides a bound on $\eta_{\mathbf{s}}(\mathbf{c}_3, \mu_1 \cdot \mu_2 \cdot \alpha)$, where α is determined solely by the evaluated circuit, and the chosen primes. This means that the multiplication algorithm effectively becomes a composition of multiplication and a multiplication by a known constant scalar α.

We now claim that every circuit can be evaluated using this modified multiplication instead of the regular one. Indeed, we take the same circuit—with the modified multiplication in place of the standard one, and associate with each cell in the circuit a scalar $\alpha_l \in \mathbb{Z}_p$. Let circuit 1 represent the circuit using the "new" multiplication and circuit 2 the one using standard multiplication. Before computation, we can determine the scalar α_l for each cell l, which is the ratio of the value of the cell in circuit 1 to the value in circuit 2 (and we need to prove that this ratio is independent of the inputs to the circuit).

Notice that when performing a homomorphic evaluation, we generally work with layered circuits, where each cell corresponds to a layer that reflects the depth of multiplications required to reach that cell. Homomorphic operations have inputs only from the same layer, as the inputs must have the same key and modulus, which are determined by the depth of multiplications. Returning to our claim, we wish to show that all the cells in the same layer have the same scalar multiplier. Indeed, we need to explain that for multiplication, addition, and scalar multiplication. For multiplication it is clear: $\alpha_l \cdot m_1 \times \alpha_l m_2 \times \alpha_{l_3} = \alpha_{l+1} \times m_1 \times m_2$ where α_{l+1} depends only on α_l, α.For addition: $\alpha_l \cdot (m_1 + m_2) = \alpha_l \cdot m_1 + \alpha_l \cdot m_2$. And for multiplication by scalar $\alpha' \cdot (\alpha_l \cdot m) = \alpha_l \cdot (\alpha' \cdot m)$.

Notice that without multiplying by scalar we have

$$\eta(\mathsf{ModSwitch}'(\mathbf{c}), \mu) \leq \frac{q'}{q}\eta(\mathbf{c}, \mu) + \hat{\gamma}\ell_1(\mathbf{s})$$

$$\eta_{\mathbf{s}_2}(\text{SwitchKey}'(\mathbf{c}, \tau_{\mathbf{s}_1 \to \mathbf{s}_2}), \mu) \leq q'/q \cdot \eta_{\mathbf{s}_1}(\mathbf{c}, \mu) + r_1 \cdot \lfloor d/2 \rfloor \cdot \lceil \log_d(q) \rceil B_{\chi_e}(q') \cdot \gamma$$

There are several ways to take advantage of this, depending on the specific usecase:

- The first and simplest approach is to always use modulus switching and key switching without multiplying by a scalar. Then, only during the final modulus switch before bootstrapping, multiply by a scalar w. Instead of choosing $w = q'/q \mod \mathfrak{p}$, choose it so that the associated scalar with the new cell becomes 1.
- Another approach is to modify the decryption algorithm to include division by the relevant scalar $\mod \mathfrak{p}$. The drawback of this method is that it cannot be applied during binary operations in bootstrapping.
- We can also attempt to influence this factor during encryption, intermediate steps, or when choosing the moduli ladder, so that the final multiplier has an inverse with a small l_1 norm.

Tradeoff between n and r. The security of the scheme depends, among other factors, on the product $n \cdot r$. When working with a ciphertext in $\mathbb{Z}/2^{n_1}$, the value of n - the degree of the polynomial can be any $n \geq n_1$. In practice, when $n_1 = 64$, we use $n > 64$ to reduce the size of the Key Switching matrix in Algorithm C.3.

D Correctness and Security of Our Bootstrapping Algorithm

The following theorem summarizes the properties of previous works that we use [KDE+24, CGGI20], as described in Section 5.1.1.

Theorem D.1 (Programmable Bootstrapping, Implicit in Prior Work). *There exist polynomial time algorithms with syntax as above, with the following properties:*

- ***Correctness.*** *Letting* $\mathsf{pbpp} = \mathsf{PB.Setup}(1^n, 1^q, \mathbf{s}, \mathbf{s}')$, $(n', q') \leftarrow \mathsf{PMap}(n, q)$, *and then computing* $\mathbf{c}' = \mathsf{PB.Bootstrap}(\mathsf{pbpp}, \mathbf{c}, F)$, *it holds that*

$$\mathbf{s}' \cdot \mathbf{c}' \pmod{q'} = F(\mathbf{s} \cdot \mathbf{c}) + e \ , \tag{5}$$

  where $|e| \leq B_{PB} = B_{\chi_\varepsilon} \cdot \mathrm{poly}(n, \log q, n', \log q')$, *for some (fixed) polynomial.*
- ***Security.*** *There exists a distribution* $\chi_{s'}$ *supported over* $\{0,1\}^{n'}$ *such that if* $\mathbf{s}' \leftarrow \chi_{s'}$ *then for all* $\mathbf{s}$

$$\mathsf{PB.Setup}(1^n, 1^q, \mathbf{s}, \mathbf{s}') \overset{c}{\approx} \mathsf{PB.Setup}(1^n, 1^q, 0, \mathbf{s}') \ ,$$

  under a cryptographic hardness assumption (Module LWE).

As a derivative of Theorem D.1, we achieve the following performance.

Theorem D.2. *Our bootstrapping procedure achieves the following performance.*

– ***Correctness.*** *Letting* $\mathsf{bspp} = \mathsf{REFHE.BS.Setup}(1^n, 1^q, \mathbf{s}, \mathbf{s}')$ *and* $(n', q') \leftarrow \mathsf{PMap}(n, q)$*, taking* $\mathbf{c} = \mathrm{REFHE}_q(\mu; \varepsilon)$ *so that* $\ell_\infty(\varepsilon) \leq q/\mathrm{poly}(n, \log q, n', \log q')$ *for some (fixed) polynomial, and then computing* $\mathbf{c}'$ *as the output of our bootstrapping procedure, it holds that* $\mathbf{c} = \mathrm{REFHE}_{q'}(\mu; \varepsilon')$ *where* $\ell_\infty(\varepsilon') \leq B_{PB} = B_{\chi_\varepsilon} \cdot \mathrm{poly}(n, \log q, n', \log q')$*, for some (fixed) polynomial.*
– ***Security.*** *There exists a distribution* $\chi_{s'}$ *supported over* $\{0,1\}^{n'}$ *such that if* $\mathbf{s}' \leftarrow \chi_{s'}$ *then for all* $\mathbf{s}$

$$\mathsf{REFHE.BS.Setup}(1^n, 1^q, \mathbf{s}, \mathbf{s}') \overset{c}{\approx} \mathsf{REFHE.BS.Setup}(1^n, 1^q, 0, \mathbf{s}') \ ,$$

under the cryptographic assumption of Theorem D.1 above in addition to the MPLWE *assumption.*

We provide a sketch of the proof below.

Security follows from Theorem D.1, in addition to the security of key-switching that follows from MPLWE.

We show correctness of the bootstrap Algorithm 5.3 by induction on the iteration index i. We claim that for each $0 \leq i < n$, c^i_{REFHE} is a REFHE encryption of $x^i \mu_i$ with fresh noise e_i, whose size is upper bounded by $|e_i| \leq (B_{PB} + B) \cdot \mathrm{poly}(n, \log(Q))$.

For $i = 0$, by the correctness of Algorithm Sample Extract 5.1, we get that $c_{\mathsf{lsb}} = \mathrm{BFV}_q(\mu_0; \varepsilon)$ is a B/FV encryption of the free coefficient μ_0. Then, by the correctness of B/FV programmable bootstrap (Theorem D.1), $\tilde{\mathbf{c}}$ consists of a vector of B/FV ciphertexts, that encrypt $\mu_0 \cdot \mathbf{g} \in \{0, \mathbf{g}\}$. Since $\tilde{\mathbf{c}}$ is the output of a bootstrap routine, its noise level is independent of the noise level of the input ciphertext c_{IN}. Next, we take the coefficient representation of $(x-2)^{-1} \pmod q \in \mathcal{R}$, and decompose it in base B to get $\mathbf{G}^{-1}_{B,Q}([\frac{1}{x-2}])$. By correctness of gadget decomposition, we have that $\mu_0 \cdot \mathbf{g} \cdot \mathbf{G}^{-1}_{B,Q}([\frac{1}{x-2}]) = [\frac{\mu_0}{x-2}]$. Therefore, $\hat{\mathbf{c}}_0 = \tilde{\mathbf{c}} \cdot \mathbf{G}^{-1}_{B,Q}([\frac{1}{x-2}])$ is vector of BFV encryptions of $[\frac{\mu_0}{x-2}]$. The noise term in each ciphertext is multiplied by $\mathcal{O}(B)$ as we instantiate the gadget decomposition with powers of B (see Definition A.3). Then, by correctness of Repacking 5.2, c^0_{MPLWE} is a BFV RLWE encryption of the polynomial $\mu \cdot (x-2)^{-1} \pmod q \in \mathcal{R}$. After multiplication by $(x-2)$, we get a REFHE ciphertext c^0_{REFHE} encrypting μ_0, with the same error term.

The next two steps in Algorithm 5.3 are responsible for subtracting the extracted bit μ_0 and prepare for the next iteration to extract the next bit μ_1 of the message. In Line (8), μ_0 is homomorphically subtracted from the input ciphertext encrypting μ. Note that a modulo switch is necessary since c^0_{REFHE} has ciphertext modulus Q that corresponds to the maximal homomorphic capacity after bootstrap. Then, in Line (9), we multiply by $x^{-1} \pmod q$ in order to shift μ_1 to be the next free coefficient term. Note that $s \cdot c' = x\mu_1 + \ldots + x^{n-1}\mu_{n-1} + x^n \cdot e$, where we use $x^n \equiv x - 2$ in $\mathcal{R}$. Therefore, after multiplying by $x^{-1} \pmod q$, we have that $s \cdot c_{\ll} = \mu_1 + \ldots + x^{n-2}\mu_{n-1} + x^{n-1} \cdot e$.

To conclude, we get that after the first iteration, $c_{\ll}$ is a homomorphic encryption of the “right shift”, $\mathsf{m} \gg 1$, of $\mathsf{m} = \mathsf{Decode}(\mu)$. We can therefore repeat this process n times, and by induction, in the i^{th} iteration, we will get that $c_{\ll}$

is a REFHE encryption of $\mu \ll i+1$, and that c^i_{REFHE} is an encryption of $x^i \mu_i$ with fresh noise. Therefore, their sum corresponds to a fresh encryption of μ as desired.

E Security of MPLWE in Our Ring

In the section we discuss the hardness of the MPLWE problem (Definition 3.2). We do so by first explaining how it relates to other algebraic variants of the Learning With Errors problems. Later we argue about the security in our specific ring, defined by polynomials of the form $f(x) = x^n - x + 2$.

E.1 Algebraic Variants of LWE and Their Relations

Algebraic variants of LWE work over a number field K. The Ring LWE (RLWE) problem [LPR10] is defined over the so-called ring of integers of K, whereas the Order LWE problem (OLWE) [BBPS19] is more general and can be instantiated over any full rank subring of the ring of integers (such subrings are called "orders", and the ring of integers itself is the maximal order). Also relevant to our work is the Polynomial LWE (PLWE) problem [SSTX09] which has a somewhat simpler definition. In PLWE, all arithmetics are done in the ring itself (rather than the "dual ring" which can be thought of as a fraction over the ring), and noise is generated in the coefficient embedding (rather than the canonical embedding). We provide some formal definitions below.

Let $f(x)$ be an irreducible monic polynomial of degree n, let $K = \mathbb{Q}[x]/\langle f(x)\rangle$ be a number field, define the ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x)\rangle$, $\mathcal{O}_K$ to be its ring of integers of the number field.

Definition E.1 (Module Order LWE). *Let $\mathcal{O}$ be an order of K and $\mathfrak{q}$ be an ideal in the order. Denoting $\mathcal{O}_{\mathfrak{q}} = \mathcal{O}/\mathfrak{q}$. Let $r \in \mathbb{N}$, χ_s a distribution over $(\mathcal{O}^\vee)^r$ and χ_ε a distribution over $\mathcal{O}^\vee$.*

For a row vector $\mathbf{s} \leftarrow \chi_s$, consider the distribution $A_{\mathbf{s}}$ over $\mathcal{O}^r_{\mathfrak{q}} \times \mathcal{O}^\vee_{\mathfrak{q}}$ defined as $\{(\mathbf{a}, \mathbf{sa} + \varepsilon \pmod{\mathfrak{q}})\}$, where $\mathbf{a}$ is uniform in $\mathcal{O}^r_{\mathfrak{q}}$, ε is sampled from χ_ε.

Then the (decisional) Module Order LWE problem (MOLWE) with respect to parameters $\mathsf{molweparams} = (f, n, \mathcal{O}, r, \mathfrak{q}, \chi_s, \chi_\varepsilon)$ is to distinguish $A_{\mathbf{s}}$ from the uniform distribution over $\mathcal{O}^r_{\mathfrak{q}} \times \mathcal{O}^\vee_{\mathfrak{q}}$, given an a-priori unbounded number of samples, where $\mathbf{s}$ is drawn from χ_s.

We point out a few important special cases:

1. *The case $\mathcal{O} = \mathcal{O}^\vee = \mathcal{O}_K$ is called Module-LWE (MLWE).*
2. *The case $r = 1$ is called Order-LWE (OLWE).*
3. *The case where both $\mathcal{O} = \mathcal{O}^\vee = \mathcal{O}_K$ and $r = 1$ is called Ring-LWE (RLWE).*

Known connections between the problems are summarized in Figure 3. The numbers on the arrows correspond to the following reductions.

[11] Implicit.

$$\begin{tikzcd}
 & \boxed{\text{MPLWE}} \arrow[d, "1,2"] \arrow[ddr, "4"] & \\
\boxed{\text{MLWE}} \arrow[ddr, "4"] & \boxed{\text{MOLWE}} \arrow[l, "3"'] \arrow[d, "4"] & \\
 & \boxed{\text{OLWE}} \arrow[d, "3"] & \boxed{\text{PLWE}} \arrow[l, "1,2"'] \\
 & \boxed{\text{RLWE}} &
\end{tikzcd}$$

Fig. 3: Reduction relations between various relevant Algebraic LWE problems. The numbering refers to the different security issues 1, 2, 3 and 4 numbered below.

1. **Coefficient vs. Canonical Sampling [RSW18].** Let X be a Gaussian random variable with expected value μ and co-variance matrix Σ and V some linear transformation. Then VX is also a Gaussian random variable with expected value $V\mu$ and co-variance matrix $V^t \Sigma V$. In particular when moving between the coefficient and canonical embedding when uses the Vandermonde matrix V_f defined by the complex roots of f. Thus in order to control the error after the transformation one has to analyze the Singular Value Decomposition (SVD) of the matrix V_f.
2. **The Dual Ring [RSW18].** Every order is a full rank lattice and thus one can define the dual of this lattice as $\mathcal{O}^\vee = \{\alpha \in K : \mathrm{Tr}_{K/\mathbb{Q}}(\alpha\mathcal{O}) \subseteq \mathbb{Z}\}$. The trick is to multiply by an element in the co-different ideal [Con09], this takes an instance from the dual ring to the ring itself. When this is done the error growth by the norm of this element. As it turns out understanding the dual ring and finding an element of small norm there is rather difficult. In [RSW18] the authors show that it is always possible to find such element in non-uniform time. Another options is to note that $f'(x)$ is always a member of the different ideal (meaning that $f'(x)^{-1}$ is in the co-different ideal)[Cor 3.5 [Con09]] which in some cases may give a decent reduction.
3. **The Order $\mathbb{Z}[x]/\langle f(x)\rangle$ vs. The Maximal Order [RSW18].** For any order $\mathcal{O}$ of K one can define the conductor ideal $\mathcal{C}_\mathcal{O} = \{x \in K : x\mathcal{O}_K \subseteq \mathcal{O}\}$. Multiplying by an element of the conductor can move a RLWE instance into an OLWE instance, again increasing the error by it's norm. In a similar way to the co–different ideal [RSW18] shows it is always possible to find a small element in the conductor or use $f'(x)$.
4. **Module to Ring Reductions [PP24].** The reduction here looks at a order $\mathcal{O}'$ of higher degree in a field extension of K'/K such that it is also a module

over the base order $\mathcal{O}$. The reduction maintains the "total degree" (The degree of the module times the degree of the ring) and works for a wide class of extension fields.

E.2 Security in Our Ring Compared to a Cyclotomic

It is a common choice to take K to be the cyclotomic field, this is done as many structural properties are known about them. This enables faster computations [LPR13b] but also simplifies some of the security assumptions. Namely cyclotomic polynomials are monogenic, meaning that $\mathcal{O}_K = \mathbb{Z}/\langle f(x)\rangle$. Furthermore as long as the defining cyclotomic number m does not admit a lot of distinct factors the transformation between the coefficient and canonical embeddings is asymptotically tame [BC22]. Lastly for power of two cyclotomic one can see that the dual ring is a scaling of the ring.

Where it comes to the polynomial $f(x) = x^n - x + 2$ we make the following notes. We verified numerically that for all $2 \leq n \leq 64$ our order $\mathcal{R}$ is indeed the ring of integers. We provide the details for $n = 32, 64$ in Section E.3.2. For larger values of n it becomes hard to verify square-freeness. However, there is no evidence that working in the ring of integers provides additional security compared to other orders. In particular we are not familiar with attacks that exploit this. In addition it admits a good transformation between the canonical and coefficient embeddings. For $n = 2^k$ this can be argued directly via Rouché's theorem similarly to the classes of polynomials describe in [RSW18]. Otherwise a heuristic argument can be made, as the polynomial has a large gap and therefore its roots are somewhat equidistributed in terms of their angle from the origin, and its roots are between 1 to $3^{\frac{1}{n}}$ with a geometric average of $2^{\frac{1}{n}}$. We calculated numerically the largest and smallest singular values up to $n = 1000$, and we see that they both scale with $\sqrt{n}$ with a small constant, see Section E.3.3 for details. For $n = 32$ and $n = 64$ we get $(\approx 4.6, \approx 15.2)$ and $(\approx 6.5, \approx 22.3)$ respectively

As for working in the dual or the ring itslef the case for our polynomial and cyclotomics are similar, in that we have to either multiply by the inverse of $f'(x)$ enlarging the error in the order of magnitude of the degree, or follow the reduction in [RSW18].

We also point out that some have argued that cyclotomic polynomials may be a worse choice in terms of security, compared to non-cyclotomics, since their structure may give raise to attacks. For example, in the context of the NTRU-Prime scheme, [BCLvV16] proposed to work with the non-cyclotomic $x^n - x + 1$ (which is quite similar to ours). They claim that working with such polynomials can also be done quite efficiently, and is less risky in terms of security.

Lastly we want to address a line of attacks on polynomial LWE using polynomials of the form $x^n + ax + b$ [EHL14, ELOS15, CIV16]. The general concept is to start with polynomials of the form $x^n + b$ for a "large" b and show it behave similarly to $x^n + ax + b$ for a "small" a. The first idea is that if $b = q - 1$ we get that $f(1) = 0 \mod q$ which in turns means that if there is a ring equation modulo both f and q it is possible to assign $x = 1$ and still obtain a valid equation. This collapses the PLWE instance into a one-dimensional instance with small noise. Notice that this attack hinges on the coefficient embedding of the noise $e(x)$ being small. Peikert [Pei16] surveys such "field dependent attacks" and concludes that they result from improper choice of noise parameters, that is enabled by pathological properties of the number field. Our interpretation of Peikert's conclusion is that so long as we add the noise "properly", i.e. in such a way that when looking at the respective dual instance, the noise is large enough, then no vulnerabilities are known. In essence this means that while we discuss the transformation between the canonical and coefficient embeddings for the security reduction, this distinction may be void for practical attacks as long as we work in the dual ring.

E.3 Properties of Our Polynomial

E.3.1 Irreducibility

Lemma E.1. *The polynomial $f(x) = x^n - x + 2$ is irreducible over the integers for $n \geq 2$.*

Proof. Notice that the complex roots of f have absolute value > 1, by the triangle inequality. assume $f(x) = h(x) \cdot g(x)$. We have that $h(0) \cdot g(0) = 2$, and since $h(0), g(0)$ are integers, one of them is of absolute value 1, with out loss of generality g. Then g must have a root of absolute value ≤ 1, a contradiction since this root is also a root of f.

E.3.2 Monogenicity It is well known that if the discriminant of a polynomial f is square free, then $\mathbb{Z}[x]/f(x)$ is the ring of integers of $\mathbb{Q}(x)/f(x)$. For a polynomial of the form $x^{2^k} - x + 2$, by theorem 4 in [GD84], the discriminant is $D = 2^{k-1} \cdot 2^{k \cdot 2^k} - (2^k - 1)^{2^k - 1}$. For $k = 64$:

$$2^{63} \cdot 2^{64 \cdot 2^{64}} - 63^{63}$$

factorizes into the following primes:

$$\begin{aligned}
&1399,\\
&315883,\\
&1054894487,\\
&1609025206302091,\\
&300524395301294803,\\
&102580173634571360137,\\
&86680176735434422018101644949061,\\
&1813131374962222709188812217190299
\end{aligned}$$

For $k = 32$:

$$2^{31} \cdot 2^{32 \cdot 2^{32}} - 31^{31}$$

factorizes into the following primes:

$$\begin{aligned}&53,\\&683,\\&1189674929,\\&72879316190189456125055364884319727806855527\end{aligned}$$

So both of the polynomials are monogenic.

E.3.3 Computing Singular Values We have computed the singular values of the polynomial $x^n - x + 2$ from $n = 4$ to $n = 1000$. See Figure 4. This suggests a $O(\sqrt{n})$ bound on the singular values, similar to the power of two case.

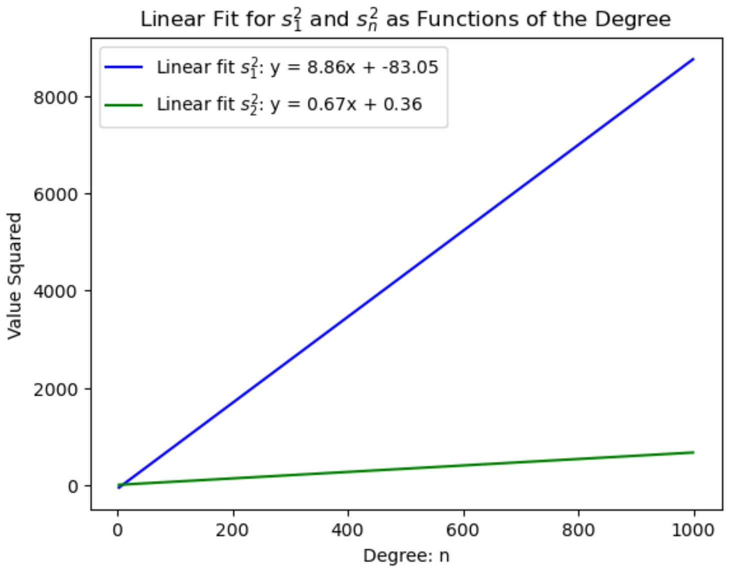


Fig. 4: Logarithmic Fit for the square of the minimal and maximal singular values as a function of the degree

F Optimizing Performance Using Ideal Ciphertext Moduli

We shortly present the scheme working with $\mathcal{R}_{\mathfrak{q}}$ where $\mathfrak{q}$ is not a rational integer. We believe it holds both theoretical and practical significance. In particular, we don't need to use multiplication by w in the key switching and modulus switching algorithms, since we can switch between input and output moduli with an element which is congruent to $1 \mod \mathfrak{p}$. Indeed there are algebraic elements which are congruent to $1 \mod \mathfrak{p}$ of much smaller ℓ_∞ norm, then rational ones. Another advantage is that we can work with ciphertexts in the "Double CRT Representation" [HS20] which means representing the ciphertext ring as a direct product of $\mathbb{Z}/n_i$. This representation allows multiplying ciphertexts without an FFT transformation to the frequency domain. Returning to our case, notice that $\mathcal{R}/\langle ax-b\rangle \cong \mathbb{Z}/(b^n - ba^{n-1} + 2a^n)$, So for cipher text modulus of the from $\mathfrak{q} = \prod \langle a_i x - b_i\rangle$, we have a double CRT representation.

In particular, we notice that for our scheme it is beneficial to set up the moduli-ladder by setting $\hat{\mathfrak{q}}_j = \langle 1 - k_j(x-2)\rangle$ for rational integers $k_j \in \mathbb{Z}$, for which $\hat{\mathfrak{q}}_j$ are coprime. We then let $\mathfrak{q}_i = q_0 \cdot \prod_{j=0}^{i} \hat{\mathfrak{q}}_j$. Here $q_0 \in \mathbb{N}$ is the "bottom step" in the ladder, which means that as we get to decrypt or bootstrap we fall back into the rational setting.

F.1 Algebraic Number Theory facts

$N(t)$ i.e. the algebraic norm of the element $t = ax - b$ (and the ideal $\langle t\rangle$ in our ring is the resultant of $ax - b$ and $x^n - x + 2$, which is $b^n - ba^{n-1} + 2a^n$. The quotient ring satisfies $\mathcal{R}/\langle t\rangle \cong \mathbb{Z}/N(t)$.

F.2 Changing the Scheme

The scheme remains structurally identical, where operations which previously performed modulo q are now conducted modulo $\mathfrak{q}$. The only aspect of the original scheme, as outlined in Algorithm 4.1, that requires further definition is the decryption algorithm. However, it is important to note that our modulus ladder ends with a rational integer. Consequently, decryption is only necessary when the modulus is a rational integer, which ensures that the scheme functions correctly without further modification.

Correctness and Security. Standard security assumptions talk about RLWE type problems with respect to a rational ciphertext modulus. Although there are assumptions with respect to alebraic ideals - as the GLWE security assumption in [PP19], we wish to talk about a simple reduction from M/P/R/O - LWE problems over an algebraic ciphertext modulus to a rational one. Working with an algebraic modulus $\mathfrak{q}$, we can perform modulus switch to the modulus $\alpha \cdot q$ for a rational q and α a polynomial with 0,1 coefficients. Notice that $\langle q\rangle | \langle \alpha \cdot q\rangle$ so in particular we get an MPLWE problem with the ciphertext modulus q - which is rational - where we lose roughly a factor of the expansion factor in the parameters.

We measure noise in ℓ_∞ in the coefficient embedding, the same as in definition B.1. Note that the analysis of the noise growth after encryption, and the bounds needed for decyption stay the same.

Homomorphic Properties. Multiplication, Multiplication by scalar and Addition are the same as in integer modulus, with q replaced by $\mathfrak{q}$ in the algorithm.

The analysis and bounds on the noise $\eta_{\mathbf{s}}(\mathbf{c}, m)$ stay the same. There difference is a new gadget in the key switching and a new modulus switching.

F.3 Representing the Ciphertext

Lemma F.1. *Assume $z \mod (ax-b)$ (for $b = 2a+1, a = k$) can be written as a polynomial of degree $n-1$ with coefficients in the range $(-b/4(1-1/2^n), b/4(1-1/2^n))$ which is $(\lceil -\frac{b}{4} \rceil, \lfloor \frac{b}{4} \rfloor)$ ($\ell_\infty(z) \leq b/4$). Then there is an efficient algorithm to find this polynomial. Also there is a unique way to write it that way.*

Proof. For (res) the resultant of $ax - b$ and $f(x)$ the generating polynomial of the number field we have:

$$z = \sum c_i (\frac{b}{a})^i \mod (\mathsf{res})$$

So

$$z \cdot a^{n-1} = \sum c_i b^i a^{n-1-i}$$

Notice that the RHS is in the interval $(-\mathsf{res}/2, \mathsf{res}/2)$. Indeed:

$$\begin{aligned} |\text{RHS}| &\leq (1 - 1/2^n) \cdot \frac{b}{4} \cdot a^{n-1} \cdot \sum \left(\frac{b}{a}\right)^i \\ &\leq (1 - 1/2^n) \cdot \frac{b}{4} \cdot a^{n-1} \cdot 2 \left(\frac{b}{a}\right)^{n-1} \\ &\leq (1 - 1/2^n) \cdot \frac{b^n}{2} \\ &\leq \frac{b^n - a^{n-1}}{2} = \frac{\mathsf{res}}{2} \end{aligned}$$

So we look at $z' = z \cdot a^{n-1} \mod \text{res}$ as an element in $\mathbb{Z}$ and provide an algorithm to write it there as a sum $\sum c_i b^i a^{n-1-i}$. Assuming such a representation exists, we must have $c_0 = z'/a^{n-1} \mod b$. Similar equation holds for $c_1 \cdot b \cdot a^{n-2}$ mod (b^2) and we can continue for the latter coefficients by taking $\mod b^i$ every time. Notice it also proves that there is a unique way to write the element as a polynomial with coefficients in this range.

Lemma F.2. *Every element t in $\mathcal{R}$ is congruent modulo $ax-b$ (where $b = 2a+1$ and $a = k$) to a polynomial z with $\ell_\infty(z) \leq \frac{b}{2} \cdot \alpha$, where $\alpha > (1 + \frac{a}{b} \cdot \zeta) \approx \frac{3}{2}$, and ζ is the real positive root of $x^n - x - 2$.*

Proof. Let $z = \sum a_i x^i$ with $a_i \in \mathbb{Z}$. Define $|z|(c)$ as the evaluation of $\sum |a_i| x^i$ at $c \in \mathbb{R}$, i.e., $|z|(c) = \sum |a_i| c^i$. Consider the polynomial z with minimal $|z|(\zeta)$ that is congruent to t modulo $ax - b$. We claim that this z cannot have coefficients with absolute value larger than $\frac{b}{2} \cdot \alpha$.

Indeed, if the i-th coefficient is larger, then we add or subtract $ax^{i+1} - bx^i$ to reduce the absolute value of the i-th coefficient (by at least $\min(b, \frac{b}{2} \cdot (2\alpha - 2))$) and claim that $|z|(\zeta)$ decreases. There are two cases to consider:

1. **Case** $i < n-1$**:** In this case, the absolute value of the $(i+1)$-th coefficient increases by at most a. If $a \cdot \zeta < \min(b, \frac{b}{2} \cdot (2\alpha - 2))$ (which holds by the assumption on α), then $|z|(\zeta)$ decreases.
2. **Case** $i = n-1$**:** The absolute value of the constant term increases by at most $2a$, and the absolute value of the x^1 coefficient increases by at most a. Therefore, $|z|(\zeta)$ decreases if

$$2a + a \cdot \zeta < \zeta^{n-1} \cdot \min(b, \frac{b}{2} \cdot (2\alpha - 2)),$$

which after dividing by ζ^{n-1} is the same inequality as in the previous case.

Working Modulo $\mathfrak{q}_j$. Recall that $\mathfrak{q}_j = q_0 \cdot \prod(a_i x - b_i)$. 1As long as the $a_i x - b_i$ are coprime (also to q_0), by CRT we have $\mathcal{R}/\mathfrak{q} \equiv \mathcal{R}/q_0 \prod \mathcal{R}/(a_i x - b_i)$, so calculations can be done coordinatewise. Notice that when doing the calculations modulo $(a_i x - b_i)$ we can choose $t \in \mathbb{Z}$ as a representative in the conjugacy class for each $c \in \mathcal{R}/(a_i x - b_i)$, So its like working in $\mathbb{Z}/(b_i^n - b_i a_i^{n-1} + 2{a_i}^n)$ in each coordinate, and then NTT can be used for multiplying elements in the quotient ring efficiently.

F.4 Key Switching

We use a variant of the Key Switching presented in algorithm C.3, with $q, q' = \mathfrak{q}, \mathfrak{q}'$ and $T = ...$ For that we introduce the following gadget:

CRT decomposition gadget Notice that we have the CRT isomorphism

$$f : \mathcal{R}_{\prod_j^l \mathfrak{q}_j} \to \prod_j^l \mathcal{R}_{\mathfrak{q}_j}$$

given that the $\mathfrak{q}_j^l$ are coprime. For $c \in \mathcal{R}_{\prod_j^l \mathfrak{q}_j}$ we look at the gadget vector $\mathbf{g}_{\text{crt}} = (f^{-1}(e_i))_i$. The decomposition operation $\mathbf{g}_{\text{crt}}^{-1} = f(c)$ viewed as a vector in $\mathcal{R}^l$. Notice that this embedding is not uniquely defined. For $\mathfrak{q}_j = ax - b$ we choose the smallest representative in absolute value in $\mathbb{Z}$ of $c \mod (ax - b)$. For $\mathfrak{q}_j = q_0 \in \mathbb{Z}$ we choose the smallest representative in ℓ_∞ in the coefficient embedding of $c \mod q_0$.

composition with powers-of-d gadget We can compose the CRT decomposition operator with the $\mathbf{g}_{\text{Powers}_d}^{-1}$ operator. The output of $\mathbf{g}_{\text{crt}}^{-1}$ is a vector when each coordinate is in $\mathbb{Z}$ with $\ell_\infty \le N(\mathfrak{q}_j)/2$ (N is the algebraic norm) or in $\mathcal{R}_{q_0}$ with $\ell_\infty \le q_0/2$. The composition is applying $\mathbf{g}_{\text{Powers}_d}^{-1}$ to each coordinate of the output meaning $\mathbf{G}_{\text{Powers}_{d_0}}^{-1} \cdot \mathbf{g}_{\text{crt}}^{-1}(x)$. Notice that applying $\mathbf{g}_{\text{Powers}_d}^{-1}$ result in different vector size in each coordinate. Notice also that this operator is the decomposition operator for the gadget $\mathbf{g}_{\text{crt}} \cdot \mathbf{G}_{powersofd_0}$.

bounding the inner product To use the Keyswitch algorithm with the powers-of-d composed to CRT decomposition gadget, we need to bound $\langle \mathbf{G}^{-1}(\mathbf{c}_1'), \mathbf{e} \rangle$ as seen in lemma C.3.

Lemma F.3. *For* $\mathbf{G}^{-1}$ *defined above, and* $\mathbf{e}$ *defined in Algorithm C.3 we have:* $\langle \mathbf{G}^{-1}(\mathbf{c}'_1), \mathbf{e}\rangle \leq \ell_\infty(\mathbf{e}) \leq \lfloor d/2 \rfloor \cdot \log_d(N(\mathfrak{q}))B_{\chi_e}(\mathfrak{q}')$

Corollary F.1. *Let* $\mathbf{c}_1 \in \mathcal{R}_{\mathfrak{q}}^{r_1}$ *and* $\mathbf{c}_2 = \text{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1)$ *defined as in algorithm C.3. Then for the gadget we defined above we have that* $\tau_{\mathbf{s}_1 \to \mathbf{s}_2}$ *is in* $\mathcal{R}_{\mathfrak{q}}^{r_1 \cdot \lceil \log_d(N(\mathfrak{q})) \rceil \times r_2}$*, and if* $\mathfrak{q}' = \mathfrak{q} \cdot \langle w \rangle$

$$\eta_{\mathbf{s}_2}(\mathbf{c}_2, \mu) \leq \gamma_w \cdot \cdot \eta_{\mathbf{s}_1}(\mathbf{c}_1, \mu) + r_1 \cdot \lfloor d/2 \rfloor \cdot \lceil \log_d(N(\mathfrak{q})) \rceil B_{\chi_e}(\mathfrak{q}') \cdot \gamma$$

F.5 Modulus Switching

We now describe the modulus switching algorithm for switching from (ideal) modulus $\mathfrak{q}$ int (ideal) modulus $\mathfrak{q}'$. Formally, we let $\mathfrak{q}, \mathfrak{q}'$ be ideals in the ring $\mathcal{R}$. We considered the special case where $\mathfrak{q} = \langle q \rangle, \mathfrak{q}' = \langle \alpha \cdot q' \rangle$ for (rational) integers q, q'.

We let $A_\mathfrak{p}$ denote a rounding algorithm for the plaintext lattice $\mathfrak{p}$ in the canonical embedding, with distance parameter $d_\mathfrak{p}$. Indeed, we always select $\mathfrak{p}$ so that it has a short basis, so such $A_\mathfrak{p}$ should exist, however its exact properties are determined by the specific instantiation we choose.

We require the existence of a *ratio parameter* $t \in K$ (we stress that t is a fraction so usually $t \notin R$) with the following properties: $t \in \mathfrak{q}' \cdot \mathfrak{q}^{-1}$ and furthermore $t = 1 \pmod{\mathfrak{p}}$. In the integer setting, if we may take $q = q' \pmod{\mathfrak{p}}$ then $t = q'/q$ is a valid ratio parameter.

Algorithm F.1: Algebraic Modulus Switching

$\mathbf{c}' \leftarrow \mathsf{ModSwitch}_{\mathfrak{q},\mathfrak{q}',t}(\mathbf{c})$, for any input $\mathbf{c} \in \mathcal{R}_{\mathfrak{q}}^{r+1}$, outputs $\mathbf{c}' \in \mathcal{R}_{\mathfrak{q}'}^{r+1}$ such that $\mathbf{c}'[i] = \lceil t \cdot \mathbf{c} \rfloor_{\mathbf{c}+\mathfrak{p}}$, as in Algorithm C.4. Notice that since the ciphertext contains multiple elements from $\mathcal{R}$, the rounding is done on each element seprately.

The following lemma summarizes the performance of the algorithm.

Definition F.1. *For* $t \in \mathbb{Q}[x]/f(x)$ *Let* $\mathcal{M}_t : \mathbb{Q}[x]/f(x) \to \mathbb{Q}[x]/f(x)$ *be the linear operator of multiplying by* t: $a \to t{\cdot}a$. *Define* $\tau(t) = \max\left\{\frac{\ell_\infty(A_t(a))}{\ell_\infty(a)} : a \in \mathbb{Q}[x]/f(x)\right\}$.

Notice that looking at $\mathcal{M}_t$ *as a matrix in the coefficient embedding,* $\tau(t)$ *is equal to the maximal* ℓ_1 *of row of the matrix*

Lemma F.4. *Let* $\mathfrak{p}, \mathfrak{q}, \mathfrak{q}', t$ *be as above. Let* $\mathbf{c} \in \mathcal{R}_{\mathfrak{q}}^{r+1}$ *be a ciphertext that encrypts* μ *under the key* $\mathbf{s}$. *Let* $\mathbf{c}' = \mathsf{ModSwitch}_{\mathfrak{q},\mathfrak{q}',t}(\mathbf{c})$. *Then*

$$\eta(\mathbf{c}', \mu) \leq \tau(t) \cdot \eta(\mathbf{c}, \mu) + \hat{\gamma} \cdot \ell_1(\mathbf{s})$$

Proof. Denote $\mathbf{s} \cdot \mathbf{c} = v + E$ where $E \in \mathfrak{q}$. By definition of $\mathsf{ModSwitch}$, it holds that

$$\mathbf{c}' = \lceil t \cdot \mathbf{c} \rfloor_{\mathbf{c}+\mathfrak{p}}^{A_\mathfrak{p}} . \tag{6}$$

Then

$$\mathbf{s} \cdot \mathbf{c}' = t \cdot v + t \cdot E + \langle \mathbf{c}' - t \cdot \mathbf{c}, \mathbf{s} \rangle \ .$$

Denote $v' = t \cdot v + \langle \mathbf{c}' - t \cdot \mathbf{c}, \mathbf{s} \rangle$, $E' = tE$. Then

$$\mathbf{s} \cdot \mathbf{c}' = v' + E'. \tag{7}$$

Since $t \in \mathfrak{q}'\mathfrak{q}^{-1}$, it holds that $E' \in \mathfrak{q}'$. Let us now analyze the norm of v'.

$$\begin{aligned} \ell_\infty(v') &= \ell_\infty(t \cdot v + \langle \mathbf{c}' - t \cdot \mathbf{c}, \mathbf{s} \rangle) & (8)\\ &\leq \ell_\infty(tv) + \ell_\infty(\langle \mathbf{c}' - t \cdot \mathbf{c}, \mathbf{s} \rangle) & (9)\\ &\leq \tau(t)\ell_\infty(v) + \hat{\gamma} \cdot \ell_1(s) \ . & (10) \end{aligned}$$

Finally, since $t \equiv 1 \mod \mathfrak{p}$, then $E' = tE \equiv E \mod \mathfrak{p}$. Furthermore we recall that $\mathbf{c}' \equiv \mathbf{c} \mod \mathfrak{p}$. Therefore

$$\begin{aligned} v' &= c_0' + \mathbf{s} \cdot \mathbf{c}' - E' & (11)\\ &\equiv c_0 + \mathbf{s} \cdot \mathbf{c} - E(\mod \mathfrak{p}) & (12)\\ &= v \ . & (13) \end{aligned}$$

This concludes the proof of the lemma.

Next we show a bound on $\tau(t)$.

Lemma F.5. *The map $A : f \to \frac{f}{1+k(x-2)}$ is reducing the ℓ_∞ norm by at least $2/3 \cdot (k+1)$, meaning $\ell_\infty(Af) \leq \frac{3}{2(k+1)} \cdot \ell_\infty(f)$*

Corollary F.2. $\tau(\frac{1}{1+k(x-2)}) \leq \frac{3}{2(k+1)}$

Proof. We upper bound the ℓ_1 norm of the rows of A as a linear operator, by $2/3 \cdot (k+1)$, which proves the lemma.

The inverse matrix of $ax + b$:

The matrix A that represents multiplying by $ax+b$ in the coefficient embedding is the following: b's on the main diagonal, a's is the diagonal bellow ($(n-2)$ a's), $A(0, n-1) = -2a$, $A(1, n-1) = a$ (indices are between 0 and $n-1$). For $n = 4$ it looks like

$$A = \begin{pmatrix} b & 0 & 0 & -2a \\ a & b & 0 & a \\ 0 & a & b & 0 \\ 0 & 0 & a & b \end{pmatrix}$$

We claim that for even n, the inverse of that matrix, is $\frac{1}{b^n + a^{n-1}b + 2a^n}$ multiplied by the following matrix B:

$$\begin{aligned} B[0,0] &= b^{n-1} + a^{n-1} \\ B[0,i] &= 2 \cdot (-b)^{i-1} a^{n-i}, && \text{for } i \geq 1 \\ B[i,i] &= b^{n-1}, && \text{for } i \geq 1 \\ B[i,j] &= (-1)^{j-i-1} \cdot (2a^{n-(j-i)} b^{j-i-1} + a^{n-1-(j-i)} b^{j-i}), && \text{for } i \geq 1 \text{and } j \geq i+1 \\ B[i,j] &= (-1)^{i-j} \cdot b^{n-1-(i-j)} a^{i-j}, && \text{for } i \geq j \text{ works also for } i = j \end{aligned}$$

For $n = 4$, the matrix B is given by:

$$B = \begin{pmatrix} b^3 + a^3 & 2a^3 & -2a^2b & 2ab^2 \\ -ab^2 & b^3 & 2a^3 + ba^2 & -(2a^2b + ab^2) \\ a^2b & -ab^2 & b^3 & 2a^3 + ba^2 \\ -a^3 & a^2b & -ab^2 & b^3 \end{pmatrix}$$

Let's see that it is indeed the inverse:

$$\begin{aligned}
(B \cdot A)[i,i] &= b \cdot B(i,i) + a \cdot B(i,i+1) \\
&= b^n + a \cdot \left(2a^{n-1} + a^{n-2}b\right), \\
&\quad \text{for } i < n-1, \\
(B \cdot A)[n-1,n-1] &= -2a \cdot (-a^{n-1}) + a \cdot a^{n-2}b + b \cdot b^n, \\
(B \cdot A)[i,n-1] &= -2a \cdot B(i,0) + a \cdot B(i,1) + b \cdot B(i,n-1) \\
&= -2a \cdot (-1)^i b^{n-1-i} a^i + a \cdot (-1)^{i-1} b^{n-i} a^{i-1} \\
&\quad + b \cdot (-1)^{n-1-i-1} \cdot \left(2a^{n-(n-1-i)} b^{n-2-i} + a^i b^{n-1-i}\right) = 0, \\
(B \cdot A)[i,j] &= b \cdot B(i,j) + a \cdot B(i,j+1) \\
&= b \cdot (-1)^{j-i-1} \cdot \left(2a^{n-(j-i)} b^{j-i-1} + a^{n-1-(j-i)} b^{j-i}\right) \\
&\quad + a \cdot (-1)^{j-i} \cdot \left(2a^{n-(j+1-i)} b^{j-i} + a^{n-1-(j+1-i)} b^{j+1-i}\right) = 0, \\
&\quad \text{for } i < j < n-1 \\
(B \cdot A)[i,j] &= b \cdot B(i,j) + a \cdot B(i,j+1) \\
&= b \cdot (-1)^{i-j} \cdot b^{n-1-(i-j)} a^{i-j} \\
&\quad + a \cdot (-1)^{i-j-1} \cdot b^{n-1-(i-j-1)} a^{i-j-1} = 0, \\
&\quad \text{for } j < n-1, j < i.
\end{aligned}$$

For $a = -k$, $b = 2k+1$ we can see that by multiplying by the inverse of A the ℓ_∞ of the vector is multiplied by at most the maximum between the ℓ_1 of the rows of the matrix, which is approximately $\frac{3}{2k}$. Indeed

$$\begin{aligned}
\ell_1 &\leq \frac{b^{n-1} + a^{n-1} + 2|a|^{n-1} + 2|a|^{n-2}b + \cdots + 2|a|b^{n-2}}{b^n + a^{n-1}b + 2a^n} \\
&= \frac{b^{n-1} + a^{n-1} + 2\frac{|a|b^{n-1} - |a|^n}{b - |a|}}{b^n + a^{n-1}b + 2a^n} \\
&\leq \frac{3}{2(k+1)}
\end{aligned}$$

Where we used that $a = -k$, $b = 2k+1$.

Corollary F.3. *Let $\mathfrak{q} = \langle a \cdot (1 + k(x-2)) \rangle$, $\mathfrak{q}' = \langle a \rangle$ and $t = \frac{1}{1+k(x-2)}$. For $\mathbf{c}, \mathbf{c}'$ as in lemma F.4, we have $\eta(\mathbf{c}') \leq \frac{3}{2(k+1)} \cdot \eta(\mathbf{c}) + \hat{\gamma} \cdot \ell_1(\mathbf{s})$*

G Recalling TFHE's Bootstrap

In this section, we recall the technique of blind rotation for bootstrapping a TFHE ciphertext. We follow the notation in [CJP21b].

The input TFHE ciphertext $c = (c_0, c_1, \ldots, c_n)$ admits $\langle c, s\rangle = c_0 s_0 + c_1 s_1 + \ldots c_n s_n = \mu^* = N/p \cdot \mu + e \mod N$, where $s_0 = -1$ and $s_i \in \{0, 1\}$, the ciphertext modulus N is a power of two, $\mu \in \frac{N}{p}\mathbb{Z}/N\mathbb{Z}$ encodes the plaintext (in $\mathbb{Z}_p$) in the most significant bits of μ^*, and the noise e is bounded by N/p, and consists of the least significant bits of μ^*.

In order to bootstrap, the rough idea is to build a *test polynomial* v, such that the coefficient of x^{μ^*} encodes the noise-free value $\frac{N}{p}\mu$ corresponding to μ^*. By homomorphically rotating the test polynomial by μ^* positions, the value of μ moves to the constant coefficient position, and it remains to extract it.

This rotation of v is done homomorphically, and since $x^{-\mu^*} \cdot v$ is a polynomial, a module LWE encryption is used during the blind rotation step of the bootstrap. Specifically, the test polynomial is encrypted under the cyclotomic ring $\mathbb{Z}[x]/\langle x^{N/2} + 1\rangle$. In this ring, x is a multiplicative element of order N, as $x^{N/2} \equiv -1$.

Therefore, since $\mu^* \equiv \langle c, s\rangle \mod N$, we have $x^{-\mu^*} = x^{-\langle c,s\rangle}$ in the $N/2$-cyclotomic ring. This observation is what enables a blind rotation of the coefficients of the test polynomial v by μ^*. Specifically, blind rotation proceeds in a sequential manner, starting from an MLWE encryption of $x^{-c_0 s_0} \cdot v = x^{c_0} \cdot v$. At each step $1 \le i < n$, the ciphertext is homomorphically multiplied by x^{-c_i} or by 1, conditioned on whether the secret key s_i. This is termed the CMux operation. It uses the i^{th} element of the bootstrapping key, which is a GSW encryption of s_i, or equivalently, an LWE encryption of the powers of B' gadget decomposition of s_i, $((B')^j s_i)_j$.

After applying the blind rotation, *sample extract* can be used to get an LWE encryption of the free coefficient.

We remark that the encryption of any other coefficient can be extracted in the same manner, and we crucially utilize this fact in our amortization technique.