Homomorphic polynomial evaluation using Galois structure and applications to BFV bootstrapping

Hiroki Okada[2], Rachel Player[1], and Simon Pohmann[1]

[1] Royal Holloway, University of London, UK
[2] KDDI Research, Japan

Abstract. BGV and BFV are among the most widely used fully homomorphic encryption (FHE) schemes. Both schemes have a common plaintext space, with a rich algebraic structure. Our main contribution is to show how this structure can be exploited to more efficiently homomorphically evaluate polynomials. Namely, using Galois automorphisms, we present an algorithm to homomorphically evaluate a polynomial of degree d in only $3\log(d)$ (in some cases only $2\log(d)$) many ciphertext-ciphertext multiplications and automorphism evaluations, where d is bounded by the ring degree. In other words, as long as the degree of the polynomial is bounded, we achieve an exponential speedup compared to the state of the art. In particular, the approach also improves on the theoretical lower bound of $2\sqrt{d}$ many ciphertext-ciphertext multiplications, which would apply if automorphisms were not available.

We investigate how to apply our improved polynomial evaluation to the bootstrapping procedure for BFV, and show that we are able to significantly improve its performance. We demonstrate this by providing an implementation of our improved BFV bootstrapping using the Microsoft SEAL library. More concretely, we obtain a 1.6× speed up compared to the prior implementation given by Chen and Han (Eurocrypt 2018). The techniques are independent of, and can be combined with, the more recent optimisations presented by Geelen *et al.* (Eurocrypt 2023).

As an additional contribution, we show how the bootstrapping approach used in schemes such as FHEW and TFHE can be applied in the BFV context. In particular, we demonstrate that programmable bootstrapping can be achieved for BFV. Moreover, we show how this bootstrapping approach can be improved in the BFV context to make better use of the Galois structure. However, we estimate that its complexity is around three orders of magnitude slower than the classical approach to BFV bootstrapping.

Keywords: homomorphic encryption, bootstrapping, implementation

1 Introduction

Fully homomorphic encryption (FHE) allows computations on encrypted data, without decrypting it or having access to the secret key. After the existence of such schemes had been an open problem for many years, Gentry [23] proposed the first FHE scheme based on lattices. Since then, much work has been done to develop more efficient and practical schemes. The BFV scheme [20] and the related BGV scheme [8], which follow from the line of work [9, 7], are among the most widely implemented today. Other FHE schemes include the TFHE family of schemes [27, 18, 17]. Notable also is the CKKS scheme [15], which is technically similar to BFV and BGV, but which differs in that it is an approximate rather than an exact scheme and in that it encrypts real numbers rather than elements in a polynomial ring.

The schemes [20, 8, 27, 18, 17, 15] all base their security on the Learning with Errors (LWE) problem [48], and hence add some noise when encrypting a message. For the exact schemes, as long as this noise is small enough, it can be removed during decryption and the exact message can be recovered. However, homomorphic operations increase the noise, and after a sufficiently long sequence of operations, the noise will exceed a certain threshold and decryption will fail. The breakthrough of [23] that allows arbitrary computations is the technique called bootstrapping, that provides a mechanism to periodically refresh the noise. The idea is simple: to perform the decryption homomorphically, using an encryption of the secret key. This way, the party performing the bootstrapping does not need the actual secret key in order to evaluate the decryption. However, it does require an additional circular security assumption that, roughly speaking, states that publishing an encryption of the secret key does not help break the scheme.

Bootstrapping for BGV and BFV has traditionally been considered together, due to the technical similarities between the two schemes. Early work approached BGV and BFV bootstrapping via boolean circuits [25]. The main idea was to represent the bootstrapping procedure using bitwise operations, making "simple" operations like multiplication quite expensive.

The algebraic structure of BGV and BFV enables their plaintext spaces to be decomposed into slots, providing single-instruction-multiple-data (SIMD) parallelism [50]. This speed up can be applied to improve bootstrapping [26]. This technique is absolutely vital for performance, and has equally been applied to present batched bootstrapping for the FHEW and TFHE scheme [45, 28].

Starting with [2], a line of work [30, 12, 22] then developed an approach for bootstrapping that avoids using boolean circuits. The main insight is that it is possible to use the algebraic structure of the plaintext space "naturally" during homomorphic decryption. In doing so, algebraic operations can directly be implemented via the corresponding homomorphic operations, vastly decreasing the overhead. However, non-algebraic operations (in particular, rounding) become more expensive, as they have to be implemented with polynomials.

In this paper, we continue the line of work [2, 30, 12, 22], and present new algebraic optimisations that can further improve the bootstrapping procedure for the BFV and BGV scheme. While we will perform our analysis only on BFV, we remark that the same techniques can be applied to BGV in a straightforward way.

1.1 Our contribution

Many BFV and BGV parameter sets used in practice have the apparent disadvantage of providing few plaintext slots of high algebraic rank, leading to a loss of parallelism. Our first main contribution is to show an advantage of the algebraic structure of these high rank slots. Namely, we show that the algebraic structure can be exploited to evaluate scalar polynomials more efficiently than is possible using generic approaches (e.g. Paterson-Stockmeyer), or with the prior methods of [47, 43]. In particular, we show how one can use the Galois structure within these high-rank slots to evaluate polynomials using only $3\log_2(d)$ key-switch operations (or in some cases even only $2\log_2(d)$ key-switch operations). This significantly improves on the previously required number of at least $2\sqrt{d}$ key switches. We apply this technique to the evaluation of the lifting polynomials used during BFV bootstrapping in power-of-two cyclotomic rings.

We remark that our techniques for improved polynomial evaluation are not just applicable to bootstrapping. Intuitively speaking, they can be used for any homomorphic evaluation of a polynomial, where the plaintext space has a high algebraic rank and the polynomial is only evaluated on elements of the prime field (respectively prime ring). As such, our results may be of wider interest.

Our second main contribution is to provide a new implementation of BFV bootstrapping in the Microsoft SEAL [49] library. This incorporates our improved approach to evaluating the lifting polynomials using the Galois structure. A previous implementation of BFV bootstrapping in SEAL has been reported [12], but the source code is not publicly available. Our performance results show that our new techniques give a notable speedup compared to the previous state of the art, being a factor of 1.6 faster than [12].

Our third main contribution is to consider the applicability to BFV of the bootstrapping techniques used in the TFHE family of schemes [27, 18, 17]. In particular, we demonstrate that so-called programmable bootstrapping, usually considered possible only in the TFHE context (see e.g. [4]), can also be achieved for BFV. Programmable bootstrapping refers to computing an arbitrary, unary function during the bootstrapping process. We propose new optimisations that again make use of the Galois structure to improve the applicability of these techniques in the BFV context. We present a theoretical comparison of a TFHE-style bootstrapping for BFV to the classical approach, which shows that it is expected to be at least three orders of magnitude slower, and so we did not implement it. Nevertheless, we expect this contribution to be of theoretical interest.

1.2 Improved polynomial evaluation

Our improvements for polynomial evaluation in BFV rely on the use of Galois theory and the field-theoretic norm $N(\cdot)$ and trace $\mathrm{Tr}(\cdot)$. As observed in previous work (see e.g. [29, 24]), we can compute them in a BFV plaintext slot

$$\mathcal{S} := \mathbb{F}_p[X]/(f), \text{ where } f \in \mathbb{F}_p[X] \text{ is irreducible,}$$

using only $2\log([\mathcal{S} : \mathbb{F}_p])$ operations. We first combine this with the fact that

$$N(\alpha - x) = \mathrm{MiPo}(\alpha)(x)$$

when α generates $\mathcal{S}$ as a ring and $x \in \mathbb{F}_p$, where $\mathrm{MiPo}(\alpha)$ the minimal polynomial of α. This enables us to evaluate irreducible polynomials of degree bounded by $[\mathcal{S} : \mathbb{F}_p]$ using only logarithmically many operations. Furthermore, this naturally extends to plaintext slots of the form $(\mathbb{Z}/p^e\mathbb{Z})[X]/(f)$, and we also study to what extent this can be used to evaluate non-irreducible polynomials.

Additionally, we introduce an alternative to the above norm-based approach, which instead uses the field trace. We discuss how the trace-based approach eliminates the need for heuristic assumptions, and has the advantage of being applicable to a broader range of problems. These include the evaluation of low-degree multivariate polynomials, especially of bilinear forms, and the evaluation of multiple polynomials in the same point. On the other hand, the trace-based approach has slightly worse performance and is more complicated that the norm-based approach.

The trace-based approach to evaluate a polynomial $g \in \mathbb{F}_p[X]$ at $x \in \mathbb{F}_p$ can be summarised as follows. First, we compute the value

$$\beta := \sum_i x^i \zeta^i,$$

where ζ is a generator of $\mathcal{S}$ with certain properties (it turns out that a root of unity works). Then, we multiply β by a constant c_g to get an element β' whose constant coefficient (i.e. coefficient of ζ^0) is $g(x)$. Finally, we use the field trace to extract this constant coefficient, thus finding $f(x)$.

1.3 Programmable bootstrapping for BFV

Programmable bootstrapping is a feature currently offered by the family of TFHE-type schemes, and refers to computing an arbitrary unary function implicitly during the bootstrapping procedure. In other words, a high-noise encryption of a message m is transformed into a low-noise encryption of $f(m)$. In TFHE-type schemes, the motivation for computing a function during bootstrapping is that the decryption algorithm requires a rounding operation. Since rounding cannot be naturally represented by algebraic operations, it is computed homomorphically using techniques for computing a generic function. Thus, replacing the rounding function by another function that additionally transforms the message does not incur additional overhead.

In BFV, this is not completely true, as an algorithm called digit extraction is known that can compute a rounding operation faster than a generic function (see Section 3.2). Nevertheless, in cases where a generic function is required to be computed, it might still be faster to combine evaluation and bootstrapping into one step. To investigate this, we try to modify the TFHE-bootstrapping procedure for BFV.

One of the most surprising ideas in TFHE bootstrapping is that, during the bootstrapping process, messages are stored in the exponent of some basis, usually a root of unity ζ. At first, this seems strange, because now even addition of messages requires an expensive homomorphic multiplication. However, this gives an advantage in the situation that we want to compute an arbitrary unary function. Namely, given an encryption of ζ^μ (where μ is the message) and a sequence a_i of elements, we find that the constant coefficient of

$$\Big(\sum a_i \zeta^{-i}\Big)\zeta^\mu$$

is a_μ. The use of this phenomena is called blind rotation, since we "rotate" the sequence a_i by the unknown message μ. When this is done, a clever key-switching technique can be used to retrieve the constant coefficient, i.e. a_μ. For details, see e.g. this excellent guide [34]. While this latter part is scheme-specific[3] , a similar approach might still have some advantages for BFV. Abstractly, an arbitrary function on the group $\langle \zeta \rangle \subseteq \mathbb{F}_{p^d}^*$ is much cheaper to compute than on $\mathbb{Z}/t\mathbb{Z}$ (assuming $t \approx \mathrm{ord}(\zeta)$), since we can exploit Galois structure.

To be somewhat more concrete, consider the following simplified scenario: Assume we have a generic function $f : \mathbb{Z}/t\mathbb{Z} \to \{0, 1\}$ that we want to evaluate on a message $\mu \in \mathbb{Z}/t\mathbb{Z}$. However, now suppose that instead of μ, we are given ζ^μ where $\zeta \in \mathbb{F}_{p^d}$ is a t-th root of unity. Then f can be computed as

$$\zeta^\mu \mapsto \sum_{y \text{ s.t. } f(y)=1} 1 - N(\zeta^\mu - \zeta^y)^{p-1},$$

where $N(\cdot)$ denotes the field-theoretic norm in $\mathbb{F}_{p^d}$. A fundamental optimization in the case of BFV would be to replace the basis (i.e. the root of unity ζ) by a primitive element of $\mathbb{F}_{p^d}$, i.e. a generator α of $\mathbb{F}_{p^d}^*$. Note that this idea does not work in TFHE, since the norm of encrypted messages in TFHE must be small to prevent huge noise growth. For BFV however, we can now choose a very small prime p (e.g. $p = 2$) and $d = \log_p(t) = \log(t)$, which results in a multiplicative depth of only $\log\log(t)$. To compare, if we wanted to compute $f(x)$ directly from x using a circuit over $\mathbb{Z}/t\mathbb{Z}$, this would require multiplicative depth $\log(t)$.

Since the repeated computation of the norm is inefficient, we also investigate other methods with slightly larger multiplicative depth that have better runtime characteristics. However, we find that all these methods for 'programmable bootstrapping' in BFV are slower than the classical approach for BFV bootstrapping by about three orders of magnitude.

1.4 Related work

Bootstrapping for the BFV/BGV family of schemes has been developed in the line of work [30, 12, 21, 22]. We especially want to highlight [21], who also

[3] It relies on the fact that in TFHE, two schemes are used. The bootstrapped scheme is based on LWE, and the "intermediate" scheme uses Ring-LWE.

proposed an optimisation for lifting polynomial evaluation during bootstrapping. As we will explain in Section 4.2, their techniques are entirely independent of ours and can also be combined, for even better performance.

Concurrent to works on BFV/BGV, there is a lot of work being done on bootstrapping in the TFHE-family. The pioneering works [3, 18] form the basis of a bootstrapping procedure that uses completely different techniques than BFV. While these schemes only bootstrap the encryption of a single bit, they are also much faster. The succeeding works [16, 17] reduced the required time to lower than 0.1 seconds. More recent work has focused more on key sizes and formats [35, 39] and decreasing the amortised bootstrapping time when bootstrapping many messages jointly [44, 45, 28, 40, 41].

An interesting hybrid in this context is [42], in which it is proposed to use BFV techniques to bootstrap many TFHE-ciphertexts in parallel. To achieve the best amortised timing, the authors of [42] consider a setting with the maximal number of slots, whereas our approach gives a speed up when there are fewer slots of high rank. Combining the two works might thus present an interesting tradeoff for amortised bootstrapping of TFHE-type ciphertexts via BFV.

Finally, there is also a line of work focusing on bootstrapping for CKKS [14, 11, 31, 32, 5, 37, 38, 36, 6]. As an approximate homomorphic scheme that performs non-exact computations on real numbers, the concept of bootstrapping in CKKS is slightly different. A high-level, non-technical comparison of different approaches to bootstrapping is given in [4].

1.5 Organisation of the paper

The remainder of the paper is organised as follows. In Section 2, we provide notation and preliminaries. In Section 3, we give an overview of the classical approach to bootstrapping, as developed in [30, 12]. In Section 4, we present our norm-based and trace-based improvements to polynomial evaluation for BFV, and discuss their application to bootstrapping. In Section 5, we report on our new implementation of BFV bootstrapping in SEAL, which includes the norm-based improvements to evaluating the lifting polynomials. In Section 6, we study the applicability of programmable bootstrapping techniques in the BFV context.

2 Preliminaries

2.1 Algebraic background

We denote by $\mathrm{MiPo}(\alpha)$ the minimal polynomial of a field extension element α. For an integer m and a prime p with $p \nmid m$, we consider the m-th cyclotomic polynomial $\Phi_m = \mathrm{MiPo}(\exp(2\pi i/m))$. Mostly, we will assume $m = 2N$ to be a power of two, and consider $\Phi_{2N} = X^N + 1$. We will consider the ring $R = \mathbb{Z}[X]/(X^N + 1)$ and its reduction modulo some prime power $R/p^e R$.

In this work, we will often require the Galois automorphism of the corresponding field extension $(R \otimes \mathbb{Q})/\mathbb{Q}$. It is a result of algebraic number theory that $R = \mathcal{O}_{R \otimes \mathbb{Q}}$ is the ring of integers in $R \otimes \mathbb{Q}$, and so there is no confusion if we denote the Galois group by $\mathrm{Gal}(R/\mathbb{Z})$. Furthermore, each Galois automorphism $\sigma \in \mathrm{Gal}(R/\mathbb{Z})$ also induces a natural automorphism

$$\sigma : R/p^e R \to R/p^e R.$$

Note that the reverse is not true, i.e. there might be automorphisms $R/p^e R \to R/p^e R$ that are not induced by a Galois automorphism of R. Hence, we will sometimes talk about the Galois group $\mathrm{Gal}((R/p^e R)/(\mathbb{Z}/p^e\mathbb{Z}))$ and mean the group of all automorphisms $R/p^e R \to R/p^e R$ that are induced by a Galois automorphism of R.

We note that modulo p, $X^N + 1$ factors into n distinct polynomials[4] of degree d, where $d = \mathrm{ord}_{(\mathbb{Z}/2N\mathbb{Z})^*}(p)$. This factorisation lifts to a factorisation modulo p^e via Hensel's lemma, and so we see that

$$R/p^e R \cong \bigoplus_{i=1}^{n} S,$$

where $S \supseteq \mathbb{Z}/p^e\mathbb{Z}$ is a free ring extension of rank d. We remark that this situation behaves almost the same as the simpler case $e = 1$, in which $S \cong \mathbb{F}_{p^d}$. In particular, there is a subgroup $G \leq \mathrm{Gal}((R/p^e R)/(\mathbb{Z}/p^e\mathbb{Z}))$ of size d that map S to S, if we fix an embedding $S \hookrightarrow R/p^e R$. We denote it by $\mathrm{Gal}(S/(\mathbb{Z}/p^e\mathbb{Z})) := G$. These are exactly the automorphisms induced by the *splitting group*

$$G_{\mathfrak{B}} = \{\sigma \in \mathrm{Gal}(R/\mathbb{Z}) \mid \sigma(\mathfrak{B}) = \mathfrak{B}\},$$

where $\mathfrak{B}$ is any prime ideal of R over p (see e.g. [46, Chapter I.9]).

2.2 The BFV scheme

The basic parameters for the BFV scheme are the *ciphertext modulus* q, the *plaintext modulus* t and the power-of-two cyclotomic ring R. In this paper, we will always have $t = p^e$, for a prime p. The plaintext space of BFV is then given as $\mathcal{P} := R/tR$ and the ciphertext space is $\mathcal{C} := (R/qR)^2$

A BFV ciphertext $(c_0, c_1) \in \mathcal{C}$ encrypts a message $\overline{m} \in \mathcal{P}$, if

$$c_0 + c_1 s = \frac{q}{t}\,\overline{m} + \epsilon,$$

where $s \in R$ is the secret key, and $\epsilon \in (R \otimes \mathbb{R})/qR$ is the noise[5], which has to be small. More precisely, for $\epsilon = \sum \epsilon_i X^i \in (R \otimes \mathbb{R})/qR$, we define the "norm" $\|\epsilon\|$ (although it is not a norm) as the ℓ_∞-norm of the smallest representative in $R \otimes \mathbb{R}$. In other words, set

$$\|\epsilon\| := \max_i \|\epsilon_i\| := \max_i \min_{\substack{z \in \mathbb{R} \\ z \equiv \epsilon_i \bmod q\mathbb{Z}}} |z|.$$

[4] Since we assume that $p \nmid m$.

[5] A short note on the algebraic structures: Clearly we have a well-defined multiplication $R \times R/qR \to R/qR$, which we can use to define $c_1 s$. Furthermore, while $(R \otimes \mathbb{R})/qR$ is not a ring anymore, it is an additive group, which suffices at this place.

This yields the following description of the BFV cryptosystem:

KeyGen: Choose $s \in R$ with coefficients s_i uniformly in $\{-1, 0, 1\}$, i.e. $\|s\| \leq 1$. Then sample $a \in R/qR$ uniformly and $\epsilon \in R/qR$ according to a discrete Gaussian distribution. Output

$$\mathrm{sk} = s, \quad \mathrm{pk} = (as + \epsilon, a).$$

Enc: To encrypt a message $\overline{m} \in R/tR$ with $\mathrm{pk} = (b, a)$, sample $\epsilon, \epsilon' \in R/qR$ from the discrete Gaussian distribution and $u \in R$ uniformly in $\{-1, 0, 1\}$ s.t. $\|u\| \leq 1$. Output

$$(c_0, c_1) = \left(bu + \epsilon + \left\lfloor \frac{q}{t} \overline{m} \right\rceil, au + \epsilon' \right).$$

Dec: To decrypt a ciphertext $(c_0, c_1) \in \mathcal{C}$ with secret key s, we compute

$$m = \left\lfloor \frac{t}{q}(c_0 + c_1 s) \right\rceil.$$

Decryption correctly recovers the message, if the noise $c_0 + c_1 s - \frac{q}{t} m$ is smaller than $\frac{q}{2t}$. Homomorphic addition of ciphertexts is achieved by summing the input ciphertexts componentwise. Homomorphic multiplication is more complicated, and we do not present the full details, referring the reader to [20]. We note that homomorphic multiplication relies on a technique called key switching.

Key switching. Key switching takes an encryption of a message m under a secret key s, and transforms it into an encryption of m under a different secret key s'. For this, a key switching key is needed, which is intuitively an encryption of s under s'.

Apart from supporting multiplication, key switching also allows us to compute the Galois automorphisms $\mathrm{Gal}(R/\mathbb{Z})$ homomorphically, since they preserve the "norm" $\| \cdot \|$. Namely, if (c_0, c_1) encrypts m and $\sigma \in \mathrm{Gal}(R/\mathbb{Z})$, we have

$$\sigma(c_0) + \sigma(c_1)\sigma(s) = \sigma(c_0 + c_1 s) = \sigma(q/t \; m + \epsilon) = \frac{q}{t}\sigma(m) + \sigma(\epsilon).$$

Therefore, $(\sigma(c_0), \sigma(c_1))$ is an encryption of $\sigma(m)$ under $\sigma(s)$. After key-switching, we then obtain an encryption of $\sigma(m)$ under s, since $\|\sigma(\epsilon)\| = \|\epsilon\|$.

Performance and noise growth. In terms of performance, key switching is the bottleneck of homomorphic computations (and thus the bottleneck in ciphertext-ciphertext multiplications and evaluation of Galois automorphisms). On the other hand, additions and plaintext-ciphertext multiplications are almost free. Because of this, we will measure the performance of different algorithms by the number of key switches that are required to compute them.

For noise growth, the picture is slightly different. In particular, ciphertext-ciphertext multiplications cause significant noise growth, while Galois automorphisms or additions cause almost negligible noise growth.

Division by p. In BFV bootstrapping, the exact division by p as

$$p(R/p^e R) \to R/p^{e-1}R, \quad x \mapsto x/p$$

is an important operation. Note that here, we change the plaintext modulus from p^e to p^{e-1}. From the definition of a BFV ciphertext, it is immediately obvious that we can implement this operation as the identity without any noise growth, if the value x is exactly divisible by p (i.e. is contained in $p(R/p^e R)$). On the other hand, if x is not exactly divisible by p, it is impossible to naturally perform the operation homomorphically.

Modulus switching. It is also possible to change the ciphertext modulus q to another ciphertext modulus q', by computing

$$(c_0', c_1') := \left(\left\lfloor \frac{q'}{q} c_0 \right\rceil, \left\lfloor \frac{q'}{q} c_1 \right\rceil \right).$$

The secret key and message remain unchanged, but some additional additive error is introduced. In this work, we only use this technique to switch to the encrypted message to a small ciphertext modulus directly before beginning the bootstrapping procedure.

3 Bootstrapping in BFV

In this section, we recall the classical approach to BFV bootstrapping [30, 12]. The BFV bootstrapping procedure is summarised in Fig. 1. Concretely, given a ciphertext $(c_0, c_1) \in \mathcal{C}$ encrypting a message in $R/p^r R$, we proceed as follows. First, we perform a modulus-switch to bring c_0, c_1 into the plaintext space $R/p^e R$ where $e > r$. The space $R/p^e R$ is sometimes called the intermediate plaintext space, as it is only used during bootstrapping. Next, we compute the noisy message $\tilde{m} = c_0 + c_1 s$. Next, we perform a linear transformation to extract the individual coefficients $\tilde{m}_i \in \mathbb{Z}/p^e\mathbb{Z}$ of $\tilde{m}$. Next, we compute the rounded division $m_i = \lfloor \tilde{m}_i / p^{e-r} \rceil \in \mathbb{Z}/p^r\mathbb{Z}$ (this is known as digit extraction). Finally, we perform the inverse linear transformation to combine the rounded coefficients m_i into the result $m \in R/p^r R$.

Slots and the linear transform. The digit extraction procedure can only ever compute the rounded quotient $\lfloor \tilde{m}_i / p^{e-r} \rceil$ for a scalar value $\tilde{m}_i \in \mathbb{Z}/p^e\mathbb{Z}$. In bootstrapping, digit extraction must be called for every coefficient of the noisy message, i.e. N times. We can use plaintext *slots* to parallelise this process, noting that if the plaintext space decomposes into a direct sum of rings $S \supseteq \mathbb{Z}/p^e\mathbb{Z}$ as

$$R/p^e R \cong \bigoplus_{i=1}^{n} S,$$

then we can perform n scalar digit extractions at once, as displayed in Fig. 3 in Appendix A. To make use of this, we need to move the coefficients of the

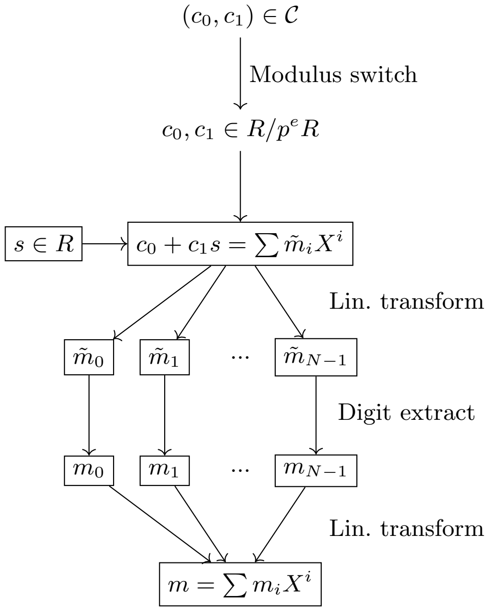

Fig. 1. The abstract bootstrapping procedure for BFV, without using slots. Each rectangle represents one ciphertext.

noisy message into the slots and back. In [26] it was shown how one can use the Galois automorphisms to do so. Commonly, this step is referred to as the *linear transformation*, even though we are just interested in one specific $\mathbb{Z}/p^e\mathbb{Z}$-linear map $R/p^rR \to R/p^rR$.

Thin/slim bootstrapping. In [12], the authors introduced the notion of "slim bootstrapping", displayed in Fig. 4 in Appendix A. This refers to the case that the message to bootstrap contains only one scalar value (i.e. value in $\mathbb{Z}/p^r\mathbb{Z}$) in each slot. In this case, we do not have to perform digit extraction on each of the N coefficients, but we only have to retrieve n scalar values. So, we can interchange the linear transform and the inner product step.

3.1 The linear transform

The basic idea underlying the linear transformation is the fact, established by [30], that any $\mathbb{Z}/p^e\mathbb{Z}$-linear transform can be written as

$$\alpha \mapsto \sum_{\sigma \in G} a_\sigma \sigma(\alpha),$$

where $a_\sigma \in R/p^eR$ and $G = \mathrm{Gal}((R/p^eR)/(\mathbb{Z}/p^e\mathbb{Z}))$. We can evaluate this somewhat efficiently using a baby-step-giant-step approach: First, we choose subsets

$G_1, G_2 \subseteq G$ of size approximately $\sqrt{N}$ such that $G_1 \cdot G_2 = G$. Then we can precompute $\sigma(\alpha)$ for all $\sigma \in G_1$ and output

$$\sum_{\tau \in G_2} \tau\Big(\sum_{\sigma \in G_1} \tau^{-1}(a_{\tau\sigma})\sigma(\alpha)\Big).$$

In our case that $R = \mathbb{Z}[X]/(X^N+1)$ is a power-of-two cyclotomic ring, no better approach is known. However, if $R = \mathbb{Z}[X]/(\Phi_m)$ is the m-th cyclotomic ring such that m has a nontrivial factorization into coprime factors $m_1, \dots, m_r$, we can do better (see e.g. [22]).

3.2 Digit Extraction

The idea of digit extraction is to write an input $x \in \mathbb{Z}/p^e\mathbb{Z}$ in base-p representation as

$$x = \sum_{i=0}^{e-1} x_i p^i, \text{ for } x_i \in \{0, ..., p-1\},$$

and give an arithmetic circuit for the “extraction function” $x \mapsto x_0$. This is done by the *lifting polynomials*, defined as follows.

Proposition 3.1 ([30]). *There exists a polynomial $f \in \mathbb{Z}[X]$ of degree at most p such that for all $1 \leq i \leq e$, $z_0 \in \{0, ..., p-1\}$, and $z_1 \in \mathbb{Z}$, have*

$$f(z_0 + p^i z_1) \equiv z_0 \mod p^{i+1}.$$

Proof. Constructed via polynomial interpolation, see Corollary 5.5 in [30]. □

In particular, a repeated application of this polynomial will “extract” the least significant digit (in base-p representation) of an input $x \in \mathbb{Z}/p^e\mathbb{Z}$. Hence, for any $x \in \mathbb{Z}/p^e\mathbb{Z}$, we find that

$$x - f^{\circ(e-1)}(x) = x - \underbrace{f(...f}_{(e-1) \text{ times}}(x))$$

is divisible by p, and quotient $(x - f^{\circ(e-1)}(x))/p$ is the result of the floor division $\lfloor x/p \rfloor$. By adding $p/2$ before doing this, we can then compute the rounded division $\lfloor x/p \rceil$.

Extracting multiple digits. Note that we want to compute the rounded division $\lfloor x/p^{e-r} \rceil$, i.e. we want to remove the $v := e - r$ least significant digits from x. Naively, we could do so by removing one digit after another, i.e., we iteratively compute

$$x^{(1)} = \frac{x - f^{\circ(e-1)}(x)}{p}, \quad x^{(2)} = \frac{x^{(1)} - f^{\circ(e-2)}(x^{(1)})}{p}, \dots, \frac{x^{(v)} - f^{\circ(r)}(x^{(v)})}{p},$$

and output $x^{(v)}$. This approach has multiplicative depth proportional to ev. To do better, note that to extract the second least-significant digit of x, it suffices to run the digit extraction procedure on $(x - f(x))/p$ instead of $(x - f^{\circ(e-1)}(x))/p$. Note that in the expression $x - f(x)$, the digits $2, \ldots, e-1$ may be altered, but the second digit is still the same as the second least-significant digit of x (and the least-significant digit is zero). This leads to a triangular computation pattern, as displayed in [12].

$$\begin{tikzcd}
x \arrow[r] & x_{01} \arrow[r] \arrow[dl] & x_{02} \arrow[r] \arrow[ddll, bend left] & x_{03} \arrow[r] & x_{04} \\
x_{10} \arrow[r] & x_{11} \arrow[r] \arrow[dl] & x_{12} \arrow[r] & x_{13} & \\
x_{20} \arrow[r] & x_{21} \arrow[r] & x_{22} & &
\end{tikzcd}$$

Fig. 2. Multiple digit extraction illustration for $e = 5$ and $r = 2$.

For example, assume $e = 5$ and $r = 2$. Then we compute the values as in Fig. 2 where $x_{ij} = x_i + p^{j+1}z$ for some $z \in \mathbb{Z}$ with the p-adic decomposition $x = \sum x_i p^i$, $x_i \in \{0, ..., p-1\}$.

Improvements for larger r. Chen and Han [12] proposed the use of a second polynomial in addition to the lifting polynomial, which can reduce the multiplicative depth in the case $r > 1$.

Proposition 3.2 (Adapted from [12, Lemma 3]). *There is a polynomial $g \in \mathbb{Z}[X]$ of degree at most $(e-1)(p-1)+1$ such that for $z_0 \in \{-\lfloor \frac{p-1}{2} \rfloor, ..., \lceil \frac{p-1}{2} \rceil\}$ and $z_1 \in \mathbb{Z}$ we have*

$$g(z_0 + pz_1) \equiv z_0 \mod p^e.$$

In other words, this directly extracts the least-significant digit of x. Or, to phrase it in terms of Fig. 2, we directly compute $x_{i0} \mapsto x_{i(e-i-1)}$. Since we require the intermediate values x_{ij} for $i + j \leq v$ to compute $x_{(i+j)0}$, we can only use this to "shorten the tail" in Fig. 2, i.e. to avoid computing x_{ij} for $v - i \leq j < e - i - 1$. However, if $r > 1$, this leads to an improvement in terms of multiplicative depth, which increases as r increases.

The work [21] characterises of all polynomials $g \in \mathbb{Z}[X]$ such that

$$g(z_0 + pz_1) \equiv z_0 \mod p^e$$

for all $z_0 \in \{-\lfloor \frac{p-1}{2} \rfloor, ..., \lceil \frac{p-1}{2} \rceil\}$. In particular, if $e > 1$, the ring $\mathbb{Z}/p^e\mathbb{Z}$ is no longer a field, and there can be many such polynomials, any of which would suffice for bootstrapping. This includes both the lifting polynomials from Prop. 3.1 and the polynomials from Prop. 3.2. Moreover, it is also shown in [21] how to find other polynomials that have lower degree or other favourable properties.

4 Evaluating polynomials

In this section, we present our improvements to the classical approach to BFV bootstrapping, as described in Section 3.

We have seen that for the digit extraction step during BFV bootstrapping, we have to homomorphically evaluate a polynomial $f \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ of degree p (or even of degree $(e-1)(p-1)+1$) at a point in $\mathbb{Z}/p^e\mathbb{Z}$. Since this polynomial is the result of interpolation, there seems to be no special structure that we can exploit for more efficient computation. Hence, we are interested in methods to evaluate a fixed, generic polynomial.

We note that efficient computations of generic polynomials (often viewed as "lookup tables") have already been discussed in the literature (e.g. [33]), and in many cases, the Galois structure is used. We will also use a Galois-based technique to efficiently compute a function $R/pR \to \mathbb{Z}/p\mathbb{Z}$ in Section 6.1. This section differs from these examples mainly in the fact that the functions we consider are over the scalar ring $\mathbb{Z}/p^e\mathbb{Z}$. At first glance, one might think that this eliminates all possibilities for using the Galois group action, but this is not the case, as we will see soon.

As ciphertext-ciphertext multiplications (as opposed to plaintext-ciphertext multiplications or additions) require key switches, they are the most expensive operations. Therefore, we seek to minimise this kind of operation. In algebraic complexity theory, this metric is also known as *nonscalar complexity* (since plaintext-ciphertext multiplications are considered scalar operations). In e.g. [10, Prop. 9.2], it is shown that the best way to evaluate a polynomial as an arithmetic circuit is the Paterson-Stockmeyer method, a variant of which was also used by [12, 21].

Paterson-Stockmeyer. In spirit, the Paterson-Stockmeyer approach uses a baby-step-giant-step idea. Assume the polynomial f of degree d is given by

$$f = \sum_{i=0}^{d} a_i X^i.$$

On input x, we can now compute the values

$$1, x, x^2, ..., x^m$$

for $m = \lceil \sqrt{d} \rceil$ using $m-1$ multiplications. Now write f as

$$f = \sum_{i=0}^{\lfloor d/m \rfloor} X^{mi} \underbrace{\sum_{j=0}^{m-1} a_{im+j} X^j}_{=: f_i} .$$

Since we have precomputed x^j for $j \leq m$, we can now evaluate all f_j at x without further ciphertext-ciphertext multiplication. After that, since we also have x^m, we can compute $f(x)$ from the $f_i(x)$ using Horner's method, i.e., as

$$f(x) = f_0(x) + x^m\Big(f_1(x) + x^m\Big(\; ... \; \Big(f_{\lfloor d/m \rfloor - 1}(x) + x^m f_{\lfloor d/m \rfloor}(x)\Big)...\Big)\Big)$$

using $\lfloor d/m \rfloor \leq m$ further multiplications. This results in nonscalar complexity $2\sqrt{d}$. We remark that this has a low nonscalar complexity, but a (relatively) high multiplicative depth $\geq \sqrt{d}$. For FHE computations, it can thus be better to use variants with slightly higher nonscalar complexity but multiplicative depth logarithmic in d.

The Paterson-Stockmeyer method is optimal for generic (more precisely, Zariski-all) polynomials *in the arithmetic circuit model*. However, in addition to additions and multiplications, we can also homomorphically compute the Galois automorphisms, and thus go beyond that model. In the rest of this section, we show how to exploit this to compute the polynomial evaluation using significantly less than $2\sqrt{d}$ multiplications/key-switches.

4.1 Using the norm

When the polynomials and the inputs are given in plain, using Galois automorphisms can already lead to small improvements, as shown in [19]. Yet their usefulness is somewhat limited by the fact that arithmetic in algebraic extension fields with nontrivial Galois group is very expensive. If Galois structure is used, it is often to reduce the situation to the scalar case. In contrast, in our situation, since the high ring degree of the plaintext/ciphertext space is necessary for security, we get the extension field arithmetic "for free". This opens up a lot of possibilities, even in the case where all inputs are already in the prime field.

In this subsection we show how to improve polynomial evaluation using the norm in the ring extension $S \supseteq (\mathbb{Z}/p^e\mathbb{Z})$ of rank $d = 2^l$. As in the field case, we define the norm of $\alpha \in S$ as

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha) := \det(m_\alpha),$$

where $m_\alpha : S \to S, \; x \mapsto \alpha x$ is the multiplication-by-α map. Then, $N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha)$ is also the constant coefficient in $\mathrm{MiPo}(\alpha)$ and

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha) = \prod_{\sigma \in \mathrm{Gal}(S/(\mathbb{Z}/p^e\mathbb{Z}))} \sigma(\alpha),$$

given that $S = (\mathbb{Z}/p^e\mathbb{Z})[\alpha]$.

The connection to polynomial evaluation is given by the next standard result of field theory.

Proposition 4.1. *Assume $S = (\mathbb{Z}/p^e\mathbb{Z})[\alpha]$. Then for $x \in \mathbb{Z}/p^e\mathbb{Z}$ have*

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha - x) = \mathrm{MiPo}(\alpha)(x).$$

Proof. If $S = (\mathbb{Z}/p^e\mathbb{Z})[\alpha]$ then also $S = (\mathbb{Z}/p^e\mathbb{Z})[\alpha - x]$, and so $\alpha - x$ also has degree d over $\mathbb{Z}/p^e\mathbb{Z}$. Now $\mathrm{MiPo}(\alpha)(T + x)$ has the root $\alpha - x$ and is of degree d, so $\mathrm{MiPo}(\alpha - x) = \mathrm{MiPo}(\alpha)(T + x)$. If we now set $\mathrm{MiPo}(\alpha) = \sum a_i T^i$, we see that

$$\mathrm{MiPo}(\alpha - x) = \sum_{i=0}^{d} a_i (T + x)^i = \sum_{i=0}^{d} \sum_{j=0}^{i} \binom{i}{j} a_i T^j x^{i-j} = \sum_{j=0}^{d} T^j \sum_{i=j}^{d} \binom{i}{j} a_i x^{i-j}$$

has the constant coefficient

$$\sum_{i=0}^{d} \binom{i}{0} a_i x^i = \mathrm{MiPo}(\alpha)(x).$$

The constant coefficient of the minimal polynomial is the norm, and the claim follows. □

As mentioned before, the norm can also be described as

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha) = \prod_{\sigma \in \mathrm{Gal}(S/(\mathbb{Z}/p^e\mathbb{Z}))} \sigma(\alpha).$$

As in the case of finite fields, we know that $\mathrm{Gal}(S/(\mathbb{Z}/p^e\mathbb{Z}))$ is cyclic of order d and generated by the Frobenius automorphism[6] $\pi : S \to S$. In the case that we are most interested in, d is a power of two, and so we can factor the norm as

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\cdot) = (\mathrm{id} \cdot \pi) \circ (\mathrm{id} \cdot \pi^2) \circ (\mathrm{id} \cdot \pi^4) \circ \ldots \circ (\mathrm{id} \cdot \pi^{d/2}).$$

This gives us an algorithm to compute the norm of an arbitrary element in S, using only $\log_2(d)$ ciphertext-ciphertext multiplications and as many Galois automorphisms. To use this for the evaluation of polynomials, we need the following additional assumption.

Heuristic 4.2. *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[T]$ be a polynomial. Then there is $g = \sum a_i X^{2^i} \in (\mathbb{Z}/p^e\mathbb{Z})[T]$ of degree less than $\deg(f)$ such that $f + g$ is irreducible.*

Note that the fraction of monic irreducible polynomials of degree r in $\mathbb{F}_p[T]$ is about $1/r$, and the corresponding result then also holds for the irreducible polynomials in $(\mathbb{Z}/p^e\mathbb{Z})[X]$. Hence, if we assume the heuristic that any given polynomial of degree r is irreducible with probability $1/r$ (and this is independent

[6] This is no longer the map $\alpha \mapsto \alpha^p$, but the unique map π that makes the diagram

$$\begin{tikzcd} S \arrow[r, "\pi"] \arrow[d, "\mathrm{mod}\ p"] & S \arrow[d, "\mathrm{mod}\ p"] \\ \mathbb{F}_{p^d} \arrow[r, "\alpha \mapsto \alpha^p"] & \mathbb{F}_{p^d} \end{tikzcd}$$

commute. It is induced by a $\sigma \in \mathrm{Gal}(R/\mathbb{Z})$ s.t. $\sigma : R/\mathfrak{B} \to R/\mathfrak{B}$ is the Frobenius, where $\mathfrak{B} \leq R$ is a prime over p.

for all polynomials), the probability that Heuristic 4.2 holds for $\deg(g) = 1$ (i.e. there is c with $f + c$ is irreducible) is about

$$1 - (1 - 1/r)^p \gtrsim 1 - \exp(-1) > 0.6$$

as wlog $r \leq p$. This probability goes very close to 1 as $\deg(g)$ increases. Thus, we find Heuristic 4.2 a reasonable assumption.

Corollary 4.3. *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[T]$ with $\deg(f) < 2^l = d$. Assume Heuristic 4.2 for a polynomial depending on f. Then we can evaluate f at $x \in \mathbb{Z}/p^e\mathbb{Z}$ using $3l$ key switches and multiplicative depth l (over S).*

Proof. Consider $f' = T^d + f(T)$ and $g \in (\mathbb{Z}/p^e\mathbb{Z})[T]$ such that $f' + g$ is irreducible. Then there is $\alpha \in S$ such that $\mathrm{MiPo}(\alpha) = f' + g$. Hence, on input $x \in \mathbb{Z}/p^e\mathbb{Z}$, we first compute the values

$$1, x, x^2, x^4, ..., x^{2^l}$$

using l multiplications. From this, we can then compute $g(x)$ without further multiplications. Finally, we compute

$$N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha - x)$$

using l multiplications and l automorphisms, and output

$$f(x) = N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha - x) - x^d - g(x). \qquad \square$$

For certain polynomials which wish to evaluate in practice, we can do even better. In particular, we assume the following stronger version of Heuristic 4.2.

Heuristic 4.4. *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ be a polynomial. Then it is quite likely that there is $c \in \mathbb{Z}/p^e\mathbb{Z}$ such that $f + c$ is irreducible.*

While Heuristic 4.4 does not hold in many cases, there are still relevant situations in which it does. In particular, bootstrapping for some parameters sets is among them, as we will illustrate in Example 4.6.

Corollary 4.5. *Let $f \in \mathbb{F}_p[T]$ monic of degree $\deg(f) = 2^l + 1$, and have $d = 2^l$. Assume Heuristic 4.4 for a polynomial depending on f. Then we can evaluate f at $x \in \mathbb{Z}/p^e\mathbb{Z}$ using $2l + 1$ key switches and multiplicative depth $l + 1$ (over S).*

Proof. Write $f = \sum_{i=0}^{d+1} a_i T^i$ and consider the polynomial

$$f' = \sum_{i=0}^{d} a_{i+1} T^i.$$

We assume there is $c \in \mathbb{Z}/p^e\mathbb{Z}$ such that $f' + c$ is irreducible, so there is $\alpha \in S$ with $\mathrm{MiPo}(\alpha) = f' + c$. Now we can compute

$$f(x) = (N_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha - x) - c) \cdot x + a_0. \qquad \square$$

To summarise, the results of this subsection show how, if a particular heuristic assumption holds, we can evaluate a polynomial f using only $2\log_2(\deg(f))$ key switch operations. This is a significant speedup compared to the $2\sqrt{\deg(f)}$ key switch operations required for the Paterson-Stockmeyer method. In Example 4.6 we show that this can be applied to the lifting polynomials during BFV bootstrapping for certain parameter sets.

Example 4.6 (Applicability of Heuristic 4.4 in the case $p = 257$). A popular parameter choice for bootstrapping is the case $p = 257$ and $e = 3$. In this case, we have $d = 256$ and $n = N/d$ for $N \geq 2^{10}$. Furthermore, the lifting polynomial $f_{\text{lift}} \in (\mathbb{Z}/257^3\mathbb{Z})[x]$ is

$$f_{\text{lift}} = x^{257} + 16941697x^{256} + 12048417x^{257} + ... + 11690673x^2 + 12066407x.$$

Since we want to use Corollary 4.5, we want to compute f_{lift}/x fast. Note here that f_{lift} has no constant term, so we can just divide out x.

By Hensel's lemma, irreducibility only depends on the value modulo p, so it suffices to consider $f := f_{\text{lift}}/x \mod 257 \in \mathbb{F}_{257}[x]$. Since irreducibility testing for polynomials over $\mathbb{F}_{257}$ is available in all major algebra packages, it is trivial to check for all $c \in \mathbb{F}_{257}$ whether $f + c$ is irreducible. Using SAGE, we find that $f + 3$ is indeed irreducible. Now we can extract a root $\tilde{\alpha} \in \mathbb{F}_{257^{256}} = S/pS$ of $f + 3$. Using the algorithmic version of Hensel's lemma, we can then lift $\tilde{\alpha} \in \mathbb{F}_{257^{256}}$ to a root $\alpha \in S := (\mathbb{Z}/257^{256}\mathbb{Z})[\zeta]$ of $f_{\text{lift}}/x + 3$. Finally, we compute the evaluation as

$$f_{\text{lift}}(x) = (N_{S/(\mathbb{Z}/257^3\mathbb{Z})}(\alpha - x) - 3) \cdot x.$$

4.2 Using the trace

In this subsection we show how to improve polynomial evaluation using the trace in the ring extension $S \supseteq (\mathbb{Z}/p^e\mathbb{Z})$ of rank $d = 2^l$. As for the norm, we define the trace as the trace of the multiplication-by-α map $m_\alpha : x \mapsto \alpha x$. As expected, it is equal to

$$\mathrm{Tr}_{S/(\mathbb{Z}/p^e\mathbb{Z})}(\alpha) = \sum_{\sigma \in \mathrm{Gal}(S/(\mathbb{Z}/p^e\mathbb{Z}))} \sigma(\alpha).$$

This avoids the heuristics needed in Section 4.1 and may be applicable to improve the evaluation of a broader class of polynomials. The trace approach has further advantages, that we describe in Section 4.3. However, to our knowledge, the trace approach cannot match the number of $2\log_2(d)$ key switches, and will thus perform worse than the norm approach described in Section 4.1. For this reason, we did not implement the trace approach.

For this subsection, we assume further that S is generated by a primitive $2N$-th root of unity ζ with N a power of two. This is exactly the case that we are typically interested in for FHE.

Lemma 4.7. *Given $x \in S$, we can compute $\alpha_1, ..., \alpha_l$ in $2l - 2$ multiplications and multiplicative depth l, where*

$$\alpha_k := \sum_{i=0}^{2^k-1} x^i.$$

Proof. We have the factorisation

$$\alpha_k = \sum_{i=0}^{2^k-1} x^i = \prod_{i=0}^{k-1} (1 + x^{2^i}).$$

Hence, we can first compute the values $x, x^2, x^4, ..., x^{2^{l-1}}$ using $l - 1$ multiplications, and then compute the product

$$(1 + x)(1 + x^2)(1 + x^4)...(1 + x^{2^{l-1}}).$$

using $l - 1$ more multiplications. Note that computing this product from left to right, we achieve multiplicative depth l and produce all the α_i, $i < l$ as intermediate results. □

Lemma 4.7 applied to $x\zeta$ allows us to jointly compute all the powers of x up to a point. This way, we find

$$\sum_{i=0}^{2^l-1} x^i \zeta^i.$$

To deduce the value of any polynomial $f(x)$ with $\deg(f) \leq 2^l - 1$ from this, it is then sufficient to compute a $\mathbb{Z}/p^e\mathbb{Z}$-linear transformation. However, if we want to be able to do this in logarithmically many key switches, we still have to choose a linear transformation that we can compute extraordinarily fast. As it turns out, the trace in the ring extension $S/(\mathbb{Z}/p^e\mathbb{Z})$ is suitable for this.

First, we prove that the minimal polynomial of ζ is sparse in a certain sense.

Lemma 4.8. $\mathrm{MiPo}(\zeta) = T^d + aT^{d/2} + b$ *for* $a, b \in \mathbb{Z}/p^e\mathbb{Z}$.

Proof. A proof is given in Supplementary Material Section B. □

Lemma 4.8 clearly implies that $\mathrm{Tr}(\zeta^k) = 0$ for $k < d$, unless $k \in \{0, d/2\}$. We want to use this to compute the linear transform

$$S \to S, \quad \sum_i a_i \zeta^i \mapsto a_0$$

using only logarithmically many Galois operations.

Lemma 4.9. *Let* $d = 2^l$ *and consider* $\alpha = \sum_{i=0}^{d-1} x_i \zeta^i \in S$. *We can compute* x_0 *and* $x_{d/2}$ *from* α *using* l *Galois automorphisms (and no nonscalar multiplications).*

Proof. Lemma 4.8 yields that $\mathrm{MiPo}(\zeta) = T^d + aT^{d/2} + b$, and so

$$\mathrm{Tr}(\zeta^k) = \begin{cases} d & \text{if } k = 0, \\ -a & \text{if } k = d/2, \\ 0 & \text{otherwise.} \end{cases}$$

Now note that $\pi(\zeta^{d/2}) = -\zeta^{d/2}$ and $\pi^{2^i}(\zeta^{d/2}) = \zeta^{d/2}$ for any $i > 0$. This means we can compute

$$\beta := \Big((\mathrm{id} + \pi^2) \circ (\mathrm{id} + \pi^4) \circ ... \circ (\mathrm{id} + \pi^{d/2})\Big)(\alpha)$$

and it takes us $l-1$ Galois automorphisms to do so. Now we have by the linearity of Tr function,

$$\begin{aligned} \mathrm{Tr}(\alpha) &= (\mathrm{id} + \pi)(\beta) = dx_0 + (-a)x_{d/2}, \text{ and} \\ \mathrm{Tr}(\alpha\zeta^{d/2}) &= (\mathrm{id} - \pi)(\beta)\zeta^{d/2} = (-a)x_0 + \underbrace{\mathrm{Tr}(\zeta^d)}_{=a^2-bd} x_{d/2}. \end{aligned}$$

Finally, we can solve the system for x_0 to find constants c_0, c_1 depending only on a, b, d such that

$$x_0 = c_0(\mathrm{id} + \pi)(\beta) + c_1(\mathrm{id} - \pi)(\beta)\zeta^{d/2}$$

and constants c_0', c_1' such that

$$x_{d/2} = c_0'(\mathrm{id} + \pi)(\beta) + c_1'(\mathrm{id} - \pi)(\beta)\zeta^{d/2}.$$

This only requires computing one more Galois automorphism, namely $\pi(\beta)$. □

Combining Lemma 4.9 with Lemma 4.7 enables us to compute polynomials.

Corollary 4.10. *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[T]$ with $\deg(f) < 2^l = d$. Assume that $\deg(f) < d/2$ or $\mathrm{MiPo}(\zeta) = T^d + a$. Then we can evaluate f at $x \in \mathbb{Z}/p^e\mathbb{Z}$ using $3l - 2$ key switches and multiplicative depth l (over S).*

Proof. First, compute

$$\alpha = \sum_{i=0}^{D-1} x^i \zeta^i$$

as in Lemma 4.7, where $D = d/2$ if $\deg(f) < d/2$ and $D = d$ otherwise. Let $f = \sum a_i T^i$. Then compute (without any further key switches)

$$\beta = \Big(\sum_{i=0}^{\deg(f)} a_i \zeta^{-i}\Big)\alpha = \Big(\sum_{i=0}^{\deg(f)} a_i \zeta^{-i}\Big)\Big(\sum_{i=0}^{D-1} x^i \zeta^i\Big) = \sum_{i=-\deg(f)}^{D-1} \zeta^i \sum_{j=0}^{D-1} a_{j-i} x^j,$$

where we set $a_i = 0$ if $i > \deg(f)$ or $i < 0$. Now the constant term of β is clearly $f(x) = \sum a_i x^i$, since by assumption no ζ^i for $i \in \{-\deg(f), ..., D-1\} \setminus \{0\}$ has any constant term. Finally, extract that constant term using Lemma 4.9. □

4.3 Advantages of the trace approach

Evaluating bivariate polynomials. In contrast to the norm-based methods, we can use the trace-based approach for evaluating a wider class of polynomials. For example, we can evaluate a bivariate polynomial $f \in (\mathbb{Z}/p^e\mathbb{Z})[X, Y]$ with bounded degree in each variable $\deg_X(f), \deg_Y(f) < 2^{\lfloor l/2 \rfloor}$. This is done by first computing

$$\alpha_1 = \sum_{i=0}^{2^{\lfloor l/2 \rfloor}-1} x^i \zeta^i \quad \text{and} \quad \alpha_2 = \sum_{i=0}^{2^{\lfloor l/2 \rfloor}-1} y^i \zeta^{2^{\lfloor l/2 \rfloor} i}$$

and then using Lemma 4.9 to extract the constant term from $c\alpha_1\alpha_2$, where $c \in S$ is an appropriate constant. While this may not be directly applicable to bootstrapping, it might still be valuable for other homomorphic operations.

Evaluating many polynomials at one point. The trace based approach is advantageous in the case that we want to evaluate multiple polynomials $f_1, ..., f_s$ of degree d in the same point x. Using a variant of Paterson-Stockmeyer, we can do this in $2\sqrt{ds}$ multiplications (using $\sqrt{ds}$ baby steps and s times $\sqrt{d/s}$ giant steps). The norm-based method still improves on that, and would give $2s\log(d)$ key switches. However, for the trace method, we only have to compute

$$\alpha = \sum_{i=0}^{d-1} x^i \zeta^i$$

once, using $2\log(d)$ multiplications. Evaluating then every $f_j(x)$ from α then only takes $\log(d)$ further key switches, thus in total $(s+2)\log(d) < 2s\log(d)$ key switches.

This situation is relevant to bootstrapping. For example, it is shown in [21] how to decrease the multiplicative depth in the case $r > 0$ by evaluating multiple digit retain polynomials in the same point x with respect to different e.

Another example of this situation is in [33], where the evaluation requires the output of a scalar function to be put into the exponent, i.e. it is required to compute

$$\tilde{f} : \mathbb{Z}/p\mathbb{Z} \subseteq R/pR \to R/pR, \quad a \mapsto \gamma^{f(a)}$$

for some function $f : \mathbb{Z}/p\mathbb{Z} \to \mathbb{Z}$. This is implemented by decomposing f into its bits, i.e. $f(x) = \sum 2^i f_i(x)$, and then computing

$$\tilde{f}(x) = \prod_i \Big(f_i(x)\gamma^{2^i} + (1 - f_i(x)) \Big)$$

so that all the functions f_i are evaluated at the same point $x \in \mathbb{Z}/p\mathbb{Z}$.

5 Implementation and results

We implemented classical BFV bootstrapping with our norm-based improvements from Section 4 in the Microsoft SEAL library [49]. We note that the

p	N	r	e	n	d	Corollary 4.5 usable?	$\log_2(q)$	Estimated levels
127	2^{15}	1	3	64	512	No	881	34/10
257	2^{15}	1	2	128	256	Yes	881	33/19

Table 1. Parameter sets used in our BFV bootstrapping implementation. The set $(p, e) = (257, 2)$ has a small chance of noise-related decryption failure. This can be mitigated either by choosing $e = 3$ and accepting fewer levels, or by using a somewhat sparse secret for reduced noise growth.

advantages of the trace-based approach discussed in Section 4 are not relevant for the application to bootstrapping, and hence we did not implement it.

To our knowledge, this is the second implementation of BFV bootstrapping in SEAL, the first one being [12]. However, their source code does not seem to be publicly available. Some of the parameter choices also differ, as explained below.

Parameters. Our performance results are given for two different parameter sets, which are summarised in Table 1. These parameter sets are similar to the ones used by [12], with the following exceptions. First, we used a uniform ternary secret distribution, instead of a sparse secret. This change affects noise growth and security. To compensate for the increased noise, we choose $e = 3$ for $p = 127$ instead of $e = 2$. Also, since sparse secrets are expected to be less secure, a smaller ciphertext modulus q (with $\log_2(q) = 806$) was used in [12], whereas we are able to use a larger q. In our implementation, the ciphertext modulus q was automatically chosen by SEAL to target a security level of 128 bits for ring dimension N, Gaussian error distribution with standard deviation $\sigma = 3.2$, and uniform ternary secret.

Settings that are out of scope. In our implementation, we only consider parameter sets with $r = 1$. In the case $r > 1$, we would also have to implement digit retain polynomials and the techniques of [21] to achieve state-of-the-art performance and noise growth. Note that these techniques can be combined with our trace-based improvements as described in Section 4.2. However, choosing the right digit retain polynomial is highly nontrivial, and we decided not to include it in our implementation.

Since our optimizations require high algebraic rank, we also do not consider parameters that provide a high number of low-rank slots, as e.g. used in [30, 13, 21]. Note that these settings require either very large plaintext moduli or non-power-of-two cyclotomic rings, both of which have significant disadvantages.

Finally, we also do not want to compare with schemes that bootstrap together with every operation. This includes the classical TFHE family of schemes [17, 18], their batched variants [44, 45, 28] and also the BFV-based bootstrapping scheme for TFHE [42]. Such a comparison is difficult, as the number of bootstrapping executions in our case is not necessarily proportional to the number of gates, but depends significantly on the shape of the evaluated circuit.

	Key switches	Time (our impl)
Slots to Coeffs	22	7.5s
Coeffs to Slots	30	7.8s
Digit Extract	63	17.0s
Total	115	32.3s

Table 2. Performance data for the $(p, e) = (127, 3)$ dataset. Since [12] uses sparse secrets, they do not consider this dataset but instead choose $(p, e) = (127, 2)$.

	Key switches	Time (our impl)	Time [12]
Slots to Coeffs	22	7.9s	-
Coeffs to Slots	30	8.6s	-
Digit Extract	17	5.6s	-
Total	69	22.1s	36.8s

Table 3. Performance data for the $(p, e) = (257, 2)$ dataset. Here we also included the timings that were reported by [12].

Performance. The implementation was tested on a system with an Intel i7-7700K 4.20GHz CPU and 16GB Ram. This system is slightly faster than the one used by [12], with an Intel i7-4770 3.40GHz CPU and 32 GB Ram. However, the difference in system speed is much smaller than our speedup, demonstrating that the new techniques can indeed improve bootstrapping performance. Both implementations are single-threaded.

The performance for slim bootstrapping with both parameter sets is displayed in Table 2 and Table 3. Note that the performance is expected to be quite sensitive to parameter choices, since the choice of prime p influences the number of slots, the degree of the lifting polynomial, and whether we can use the improved variant of norm-based evaluation given by Corollary 4.5. While [12] does not report timings for the parameter set $(p, e) = (127, 3)$, we do not expect our implementation to yield a significant speedup here. The reason is that we decrease the number of key-switches during digit extraction from about $2\sqrt{p}$ to $3\log_2(p)$, but

$$2\sqrt{127} \approx 23 \quad \text{and} \quad 3\log_2(127) \approx 21.$$

The situation is different for $p = 257$, not only due to the larger difference between $\sqrt{p}$ and $\log_2(p)$ in this case, but also since this parameter set supports the improved version of Corollary 4.5 and so costs only $2\log_2(p)$ key switches. We see from Table 3 that our norm-based improvements result in a speed up of $1.6\times$ compared to the prior reported implementation [12].

Source code. The source code is available at our github repository[7]. Our implementation provides a convenient interface to compute arbitrary linear trans-

[7] https://github.com/FeanorTheElf/galois-bootstrapping

formations efficiently. However, the interface to use the advanced polynomial evaluation techniques currently requires complicated precomputations of the polynomials. To present the implementation results in this paper, these precomputations were done using SAGE and the results were hardcoded.

6 TFHE-style programmable bootstrapping for BFV

In this section, we depart from the classical approach to BFV bootstrapping, as described in Section 3, and as improved upon in Section 4. Instead, we consider a new approach to bootstrapping BFV using techniques from the literature on TFHE bootstrapping. As a main feature, this enables us to demonstrate programmable bootstrapping for BFV in Algorithm 1. For full details of the TFHE bootstrapping procedure we refer the reader to the excellent guide [34].

6.1 A TFHE-style bootstrapping approach for BFV

Recall that BFV decryption is given as

$$m = \left\lfloor \frac{t}{q} \left(c_1 s + c_0 \right) \right\rceil \in R/tR,$$

where the ciphertext ring is R/qR and the plaintext ring is R/tR, where R is generated by a $2N$-th primitive root of unity ζ. Before evaluating decryption homomorphically, we will perform a modulus switch such that $c_0, c_1 \in R/TR$, where $T < q$ is a ciphertext modulus as small as possible, without exceeding the noise threshold. Then, we are left instead with evaluating the expression

$$m = \left\lfloor \frac{t}{T} \left(c_1 s + c_0 \right) \right\rceil \in R/tR\,.$$

While we still work in the ring R, assume for now that our message is scalar, i.e. $m \in \mathbb{Z}/t\mathbb{Z}$. Thus, the first step is to compute the noisy message

$$\tilde{m} = c_{00} + c_{10} s_0 - \sum_{i=1}^{N-1} c_{1i} s_{N-i} \in \mathbb{Z}/q\mathbb{Z},$$

the constant coefficient of $c_0 + c_1 s$, where

$$c_i = \sum c_{ij} X^j \quad \text{and} \quad s = \sum s_j X^j .$$

As in TFHE bootstrapping, we do not directly compute $\tilde{m}$ but instead $\alpha^{\tilde{m}}$ where $\alpha \in S = \mathbb{F}_{p^d}$ is a primitive element of one degree-d slot S. In particular, this allows us to take a very small bootstrapping plaintext modulus $p < t$ during bootstrapping, since we only require that $T \leq p^d$. While a small plaintext modulus decreases noise, this now requires a vast number of multiplications. Namely, we have to compute

$$\alpha^{\tilde{m}} = \alpha^{c_0} (\alpha^{s_0})^{c_{10}} \prod_{i=1}^{N-1} (\alpha^{-s_i})^{c_{1(N-i)}} .$$

Usually, we would do this in $N\log(T)$ multiplications, by providing encryptions of $\alpha^{\pm s_i 2^j}$ in the bootstrapping key (for all j). After we homomorphically compute $\alpha^{\tilde{m}}$, we then can proceed to find

$$f(\lfloor t/T \cdot \tilde{m} \rceil) \in \{0, 1, ..., p-1\}.$$

Here, we assume that $f : \mathbb{Z}/t\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ has outputs in $\mathbb{Z}/p\mathbb{Z}$. As in TFHE, we do this by a "homomorphic lookup table", which in our case could correspond to computing

$$f(\lfloor t/T \cdot \tilde{m} \rceil) = \sum_{y \in \mathbb{Z}/p\mathbb{Z}} y \sum_{\substack{x \in \mathbb{Z}/T\mathbb{Z} \text{ s.t.} \\ f(\lfloor t/T \cdot x \rceil) = y}} 1 - N(\alpha^x - \alpha^{\tilde{m}})^{p-1}.$$

Using the fact that $N(x) = \prod_\sigma \sigma(x)$ where σ runs through the Galois automorphisms of $\mathbb{F}_{p^d}$, we can expand this and find

$$\sum_{y \in \mathbb{Z}/p\mathbb{Z}} y \sum_{\substack{x \in \mathbb{Z}/T\mathbb{Z} \text{ s.t.} \\ f(\lfloor t/T \cdot x \rceil) = y}} 1 - N(\alpha^x - \alpha^{\tilde{m}})^{p-1} = G(\sigma_0(\alpha^{\tilde{m}}), \sigma_1(\alpha^{\tilde{m}}), ..., \sigma_{d-1}(\alpha^{\tilde{m}}))$$

for some polynomial $G \in \mathbb{F}_{p^d}[X_0, ..., X_{d-1}]$ of degree $p-1$ in each variable, and $\mathrm{Gal}(\mathbb{F}_{p^d}/\mathbb{F}_p) = \{\sigma_0, ..., \sigma_{d-1}\}$. Naively, we can already use this to compute the result $f(\lfloor t/T \cdot \tilde{m} \rceil)$, but it requires p^d key-switches. To do better, write

$$G = \sum_I X_0^{I_0} ... X_{d/2-1}^{I_{d/2-1}} \cdot G_I(X_{d/2}, ..., X_{d-1}),$$

where I runs over $\{0, ..., p-1\}^{d/2}$ and the G_I are multivariate polynomials in $d/2$ variables, and degree $p-1$ in each variable. On input $(x_0, ..., x_{d-1})$, we can now precompute all monomials of the form

$$m_I = x_0^{I_0} ... x_{d/2-1}^{I_{d/2-1}} \quad \text{and} \quad m'_J = x_{d/2}^{J_0} ... x_{d-1}^{J_{d/2-1}}.$$

Now each $G_I(x_{d/2}, ..., x_{d-1})$ is just a linear combination of the m'_J, and so we find

$$G(x) = \sum_I m_I G_I(x_{d/2}, ..., x_{d-1})$$

using just $3p^{d/2}$ key-switches. The multiplicative depth is very low, namely $\log(p-1) + \log(d)$. The bootstrapping process is summarised in Algorithm 1.

Caveats. While Algorithm 1 can be implemented, we made some assumptions that might be inconvenient in practice. First and foremost, we assumed that our unary function $f : \mathbb{Z}/t\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ has images in $\mathbb{Z}/p\mathbb{Z}$. While we can easily homomorphically compute the map $\mathbb{Z}/p\mathbb{Z} \to \mathbb{Z}/t\mathbb{Z}$ if $p \mid t$, this still limits the amount of possible output values.

Furthermore, we assumed that the message m is scalar, i.e., an element of $\mathbb{Z}/t\mathbb{Z}$. In practice, we are interested in full messages that are any element of

Algorithm 1: TFHE-style programmable bootstrapping for BFV.

```
Input: Ciphertext $(c_0, c_1) \in (R/qR)^2$ encrypting scalar message $m \in \mathbb{Z}/t\mathbb{Z}$;
       Bootstrapping keys $(k_0^{(i,j)}, k_1^{(i,j)})$ encrypting $\alpha^{s_0 2^j}$ resp. $\alpha^{-s_i 2^j}$;
       Decomposition of $G = \sum_I X_0^{I_0} ... X_{d/2-1}^{I_{d/2-1}} G_I(X_{d/2}, ..., X_{d-1})$
Output: Output $(c'_0, c'_1) \in (R/qR)^2$ encrypting scalar message $f(m) \in \mathbb{Z}/p\mathbb{Z}$
        with low noise
Modulus switch $(c_0, c_1)$ to $(\tilde{c}_0, \tilde{c}_1)$ in $R/TR$
Decompose $\tilde{c}_1$ as $\sum_i \sum_j c_{ij} 2^j X^i$ with $c_{ij} \in \{0, 1\}$
Compute ciphertext ct as homomorphic evaluation of
    $$\tilde{m} := \alpha^{\tilde{c}_0} \prod_j (\alpha^{s_0 2^j})^{c_{0j}} \prod_{i>0,j} (\alpha^{-s_{N-i} 2^j})^{c_{ij}}$$
Consider tables $\mathrm{ct}[I]$, $\mathrm{ct}'[I]$ of ciphertexts where $I \in \{0, ..., p-1\}^{d/2}$
for $i \in \{0, ..., d/2-1\}$ do
    Compute $\mathrm{ct}[e_i]$ as homomorphic evaluation of the Galois conjugate $\sigma_i(\tilde{m})$
      ($e_i \in \{0, ..., p-1\}^{d/2}$ is the unit vector)
    Compute $\mathrm{ct}'[e_i]$ as homomorphic evaluation of $\sigma_{i+d/2}(\tilde{m})$
end
for $I \in \{0, ..., p-1\}^{d/2} \setminus \{e_i\}$ ordered by ascending $\ell_1$-norm do
    Compute $\mathrm{ct}[I]$ as encrypted product of $\mathrm{ct}[\lfloor I/2 \rceil]$ and $\mathrm{ct}[I - \lfloor I/2 \rceil]$
    Compute $\mathrm{ct}'[I]$ as encrypted product of $\mathrm{ct}'[\lfloor I/2 \rceil]$ and $\mathrm{ct}'[I - \lfloor I/2 \rceil]$
    (During rounding in $\lfloor I/2 \rceil$, assume we round up respectively down
      alternately)
end
for $I \in \{0, ..., p-1\}^{d/2}$ do
    Compute homomorphic evaluation of $g_I := \sum_J a_J m'_J(\sigma_{d/2}\tilde{m}, ..., \sigma_{d-1}\tilde{m})$
      where $G_I = \sum_J a_J m'_J$ as linear combination of the $\mathrm{ct}'_J$
    Compute $\mathrm{ct}''_I$ as encrypted product of $g_I$ and $m_I(\sigma_0\tilde{m}, ..., \sigma_{d/2-1}\tilde{m})$
end
Output $\sum_I \mathrm{ct}''_I$
```

R/tR, or at least in messages containing one scalar in each slot. The only way Algorithm 1 can be used to bootstrap many scalars is to use slots for SIMD parallelism. However, this conflicts with the choice of a very small intermediate plaintext modulus p (such as $p = 2$ or $p = 3$). Apart from performance, such a choice can also greatly reduce the noise cause by BFV multiplications, as the growth is proportional to the plaintext modulus.

Complexity. We now give a theoretical analysis of the complexity of TFHE-style BFV bootstrapping. The first step is to homomorphically compute $\alpha^{\tilde{m}}$ using $N \log(T)$ multiplications and multiplicative depth $\log(N) + \log\log(T)$. Then, we perform the "homomorphic lookup table" step to compute $f(\lfloor t/T\tilde{m} \rceil)$, using $3p^{e/2}$ key-switches and multiplicative depth $\log(p-1) + \log(e)$. Up to now, we implicitly assumed $e = d$, but by choosing α to be a generator of a Galois subring

		Classical	TFHE-style (Algo. 1)	Both
e	Exponent		$e \mid d$	
t	Input plaintext modulus	$t = p$		
T	Intermediate plaintext modulus	$T = p^e$	$T = p^e - 1$	$T \geq 4Nt$
d, n	Slot rank, slot count			$N = nd$, $d = \mathrm{ord}(p)$
$\mathcal{P}$	Input plaintext space			$\mathcal{P} = R/tR$
$\mathcal{P}'$	Bootstrapping plaintext space	$\mathcal{P}' = R/TR$	$\mathcal{P}' = R/pR$	

Table 4. Parameter constraints for BFV bootstrapping approaches.

of a slot, we can instead work with any $e \mid d$. Hence, the total number of key switches is

$$N \log(T) + 3p^{e/2} = (1 + o(1)) \left(N \log(T) + 3\sqrt{T} \right).$$

Furthermore, the total multiplicative depth is

$$\lceil \log(N) + \log\log(T) \rceil + \lceil \log(p-1) \rceil + \log(e) \leq \log(N \log(T)(p-1)e) + 2.$$

Since the error per multiplication in BFV is $(2 + o(1))(pN^2)$ [20, Lemma 2], this yields a total bootstrapping error of

$$(1+o(1)) \log(pN^2)(\log(N \log(T)(p-1)e)+2) = (1+o(1)) \log(pN^2) \log(Npe \log(T))$$

bits. If we assume that p is a small constant, we find $e = O(\log(T))$ and so the expression of the error simplifies to

$$(2 + o(1)) \log(N) \log(N \log(T)^2) = (1 + o(1)) \left(2 \log(N)^2 + 4 \log(N) \log\log(T) \right).$$

For parameters used in practice, this is slightly lower than the noise of standard bootstrapping, mainly due to the choice of a very small plaintext modulus.

6.2 Comparison with classical BFV bootstrapping

In this subsection, we give a theoretical comparison of the classical approach to BFV bootstrapping with our alternative, TFHE-style approach. The comparison shows that the TFHE-style approach would be significantly slower than the classical approach, and so we did not implement it.

Parameters. In order to compare the both the classical and the TFHE-style bootstrapping approaches, we need to establish a set of shared parameters that we can use in the analysis. We summarise the constraints on the parameters in Table 4.

	Classical (slim) bootstrapping	TFHE-style bootstrapping (Algorithm 1)
Input scalar message	$m \in \mathcal{P}$, sparsely packed	$m \in \mathbb{Z}/t\mathbb{Z} \subseteq \mathcal{P}$
Total key-switches*	$4\sqrt{n} + \frac{3}{2}\log(p)(e-1)^2$	$N\log(T) + 3p^{e/2}$
if $T \approx p^e$, p, n constant	$\approx \frac{3}{2}\log(T)^2$	$\approx N\log(T) + 3\sqrt{T}$
Multiplicative depth	$\leq (e-1)\log(p) + 2$	$\leq \log(N\log(T)(p-1)e) + 2$
if $T \approx p^e$, p constant	$= (1+o(1))\log(T)$	$= (1+o(1))\log(N\log(T)^2)$
Bootstrapping noise bits*	$\log(T)\log(N^2T)$	$\log(Npe\log(T))\log(N^2p)$
if $T \approx p^e$, p constant	$= \log(T)^2 + 2\log(T)\log(N)$	$\approx 2\log(N)^2 +$ $4\log(N)\log\log(T)$

Table 5. An estimate of the performance of classical (slim) bootstrapping (as described in Section 3 and Section 4, using sparsely packed plaintexts) and TFHE-style programmable bootstrapping (as described in Section 6.1). Metrics marked with * are only given up to a factor of $(1 + o(1))$.

Results. In Table 5 we give a performance comparison of both bootstrapping approaches. The asymptotic values for the TFHE-style bootstrapping are as explained in Section 6.1. For classical bootstrapping, the asymptotics are obtained by an analogous, straightforward computation, and include the improvements to classical bootstrapping that we have presented in Section 4.

In Table 6 we then give an example of concrete parameters for each bootstrapping approach, and corresponding concrete costs according to Table 5. The concrete values are obtained by evaluating the asymptotic expressions, ignoring the $(1 + o(1))$ factor. The security estimates were computed using the Lattice Estimator of [1].

We remark that our estimates of the noise relies on a *worst-case* analysis. An average noise noise analysis may enable us to choose a smaller ciphertext modulus q. This in turn improves security, which then might allow smaller ring dimension N. However, we believe that the results of the comparison would not look significantly different under an average-case analysis, hence we are satisfied with the simpler, worst-case estimates.

The "total complexity" metric of Table 6 gives a basic approximation of the total number of operations required for the computation. We still only count key switches, but each key switching operation performs

$$c\log(q)N\log(N), \quad c \text{ constant}$$

constant-size arithmetic instructions where $c > 0$ is some constant, using either the base-decomposition method, or an RNS implementation. We expect the time required for each approach to be proportional to the estimate of the total complexity given in Table 6. That is, we estimate that the TFHE-style bootstrapping would be about $2^{10} \approx 1000$ times slower than a classical BFV bootstrapping, and thus entirely impractical.

	Classical (slim) bootstrapping	TFHE-style bootstrapping (Algorithm 1)
N	2^{16}	2^{15}
p	257	3
T	257^3	$3^{16} - 1$
t	257	257
(n, d)	$(64, 2^{10})$	$(4, 2^{13})$
Key switches	$1.4 \cdot 10^2$	$8.5 \cdot 10^5$
Noise (worst-case, bits)	1345	798
Ciphertext mod q (bits)	1410	840
Security (bits)	128	100
Total complexity	$2^{38} \cdot c$	$2^{48} \cdot c$

Table 6. Example concrete parameters for both bootstrapping approaches, with an estimate of their security, and an estimate of their complexity, based on Table 5. The parameters are chosen based on a simple worst-case noise analysis, hence they do not match the parameters used in our implementation in Section 5.

References

[1] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015).

[2] Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 1–20. Springer (2013).

[3] Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 297–314. Springer International Publishing, Berlin, Heidelberg (2014)

[4] Badawi, A.A., Polyakov, Y.: Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149 (2023), `https://eprint.iacr.org/2023/149`

[5] Bossuat, J., Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology – EUROCRYPT 2021, Proceedings, Part I. pp. 587–617. Springer (2021)

[6] Bossuat, J., Troncoso-Pastoriza, J.R., Hubaux, J.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security ACNS 2022. pp. 521–541. Springer (2022)

[7] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 868–886. Springer (2012).

[8] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Paper 2011/277 (2011), `https://eprint.iacr.org/2011/277`

[9] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 505–524. Springer International Publishing, Berlin, Heidelberg (2011)

[10] Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic complexity theory, vol. 315. Springer Science & Business Media (2013)

[11] Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 34–54. Springer (2019).

[12] Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 315–337. Springer (2018).

[13] Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. Cryptology ePrint Archive, Paper 2018/067 (2018), `https://eprint.iacr.org/2018/067`

[14] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 360–384. Springer (2018).

[15] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer (2017).

[16] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 3–33. Springer International Publishing, Berlin, Heidelberg (2016)

[17] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020).

[18] Ducas, L., Micciancio, D.: Fhew: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 617–640. Springer International Publishing, Berlin, Heidelberg (2015)

[19] Elia, M., Rosenthal, J., Schipani, D.: Polynomial evaluation over finite fields: new algorithms and complexity bounds. Applicable Algebra in Engineering, Communication and Computing **23**(3-4), 129–141 (2012)

[20] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), `https://eprint.iacr.org/2012/144`

[21] Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 257–286. Springer Nature Switzerland (2023).

[22] Geelen, R., Vercauteren, F.: Bootstrapping for BGV and BFV revisited. Journal of Cryptology **36**(2), 12 (3 2023).

[23] Gentry, C.: A fully homomorphic encryption scheme. Stanford university (2009)

[24] Gentry, C., Halevi, S., Peikert, C., Smart, N.P.: Field switching in BGV-style homomorphic encryption. Cryptology ePrint Archive, Paper 2012/240 (2012). , `https://eprint.iacr.org/2012/240`

[25] Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2011/680 (2011), `https://eprint.iacr.org/2011/680`

[26] Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 465–482. Springer (2012).

[27] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 75–92. Springer International Publishing, Berlin, Heidelberg (2013)

[28] Guimarães, A., Pereira, H.V.L., van Leeuwen, B.: Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. Cryptology ePrint Archive, Paper 2023/014 (2023), https://eprint.iacr.org/2023/014

[29] Halevi, S., Shoup, V.: Algorithms in HElib. In: Advances in Cryptology – CRYPTO 2014. pp. 554–571. Springer (2014)

[30] Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (2021)

[31] Han, K., Hhan, M., Cheon, J.H.: Improved homomorphic discrete fourier transforms and FHE bootstrapping. IEEE Access **7**, 57361–57370 (2019)

[32] Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Jarecki, S. (ed.) Topics in Cryptology - CT-RSA 2020. pp. 364–390. Springer (2020)

[33] Iliashenko, I., Izabachène, M., Mertens, A., Pereira, H.V.: Homomorphically counting elements with the same property. Proceedings on Privacy Enhancing Technologies **4**, 670–683 (2022)

[34] Joye, M.: Guide to fully homomorphic encryption over the [discretized] torus. Cryptology ePrint Archive, Paper 2021/1402 (2021), https://eprint.iacr.org/2021/1402

[35] Joye, M., Paillier, P.: Blind rotation in fully homomorphic encryption with extended keys. In: Dolev, S., Katz, J., Meisels, A. (eds.) Cyber Security, Cryptology, and Machine Learning. pp. 1–18. Cham (2022)

[36] Jutla, C.S., Manohar, N.: Sine series approximation of the mod function for bootstrapping of approximate HE. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Proceedings, Part I. pp. 491–520. Springer (2022)

[37] Lee, J., Lee, E., Lee, Y., Kim, Y., No, J.: High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology – EUROCRYPT 2021, Proceedings, Part I. pp. 618–647. Springer (2021)

[38] Lee, Y., Lee, J., Kim, Y., Kim, Y., No, J., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Proceedings, Part I. pp. 551–580. Springer (2022)

[39] Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Proceedings, Part III. pp. 227–256. Springer (2023)

[40] Liu, F., Wang, H.: Batch bootstrapping I: - A new framework for SIMD bootstrapping in polynomial modulus. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Proceedings, Part III. pp. 321–352. Springer (2023)

[41] Liu, F., Wang, H.: Batch bootstrapping II: - bootstrapping in polynomial modulus only requires $\tilde{o}(1)$ FHE multiplications in amortization. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Proceedings, Part III. pp. 353–384. Springer (2023)

[42] Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7ms, with $\tilde{O}(1)$ polynomial multiplications. Cryptology ePrint Archive, Paper 2023/910 (2023), https://eprint.iacr.org/2023/910

[43] Maeda, D., Morimura, K., Narisada, S., Fukushima, K., Nishide, T.: Efficient homomorphic evaluation of arbitrary uni/bivariate integer functions and their applications. Cryptology ePrint Archive, Paper 2023/366 (2023). , https://eprint.iacr.org/2023/366

[44] Miccianco, D., Sorrell, J.: Ring packing and amortized FHEW bootstrapping. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) ICALP 2018. pp. 100:1–100:14 (2018).

[45] Micheli, G.D., Kim, D., Micciancio, D., Suhl, A.: Faster amortized FHEW bootstrapping using ring automorphisms. Cryptology ePrint Archive, Paper 2023/112 (2023), https://eprint.iacr.org/2023/112

[46] Neukirch, J.: Algebraic number theory, vol. 322. Springer Science & Business Media (2013)

[47] Okada, H., Cid, C., Hidano, S., Kiyomoto, S.: Linear depth integer-wise homomorphic division. In: WISTP 2018. pp. 91–106. Springer (2019)

[48] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of STOC '05. p. 84–93. Association for Computing Machinery (2005)

[49] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (1 2023), microsoft Research, Redmond, WA.

[50] Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**, 57–81 (2014)

A BFV bootstrapping Figures omitted from Section 3

Bootstrapping with slots, resp. thin bootstrapping is shown in Fig. 3, resp. Fig. 4.

B Proof of Lemma 4.8

Proof. First, we find a number field extension $L/K/\mathbb{Q}$ such that. L has a primitive $2N$-th root of unity ξ with

$$\mathrm{MiPo}_K(\xi) \equiv \mathrm{MiPo}(\zeta) \mod p^e.$$

To do so, consider a maximal ideal $\mathfrak{B}$ over (p) of R. Then $R/\mathfrak{B}^e \cong S$ and so we can use Hensel's lemma to find a primitive $2N$-th root of unity ξ in the completion $R_{\mathfrak{B}}$ with $\xi \equiv \zeta \mod \mathfrak{B}^e$.

Therefore $L := \mathbb{Q}[\xi]$ is now the $2N$-th cyclotomic number field, so $L = R \otimes \mathbb{Q}$ and $R = \mathcal{O}_L$ is the ring of integers of L. Now define the splitting group

$$G := \{\sigma \in \mathrm{Gal}(L/\mathbb{Q}) \mid \sigma(\mathfrak{B}) = \mathfrak{B}\}$$

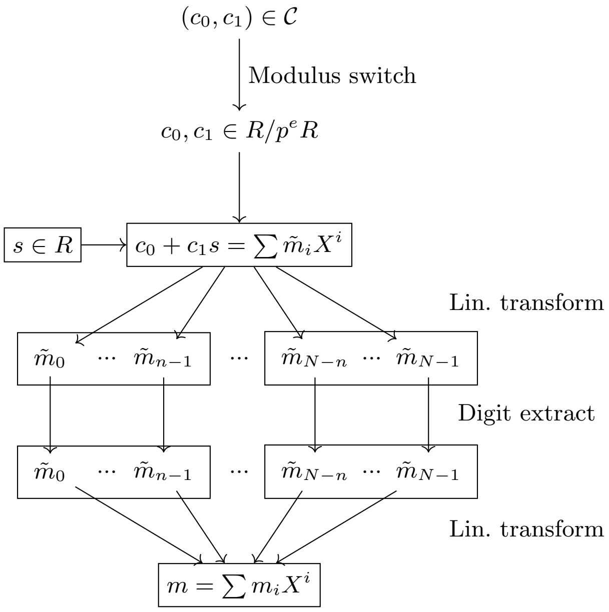
$(c_0, c_1) \in \mathcal{C}$
Modulus switch
$c_0, c_1 \in R/p^e R$
$s \in R$
$c_0 + c_1 s = \sum \tilde{m}_i X^i$
Lin. transform
$\tilde{m}_0$ ··· $\tilde{m}_{n-1}$
···
$\tilde{m}_{N-n}$ ··· $\tilde{m}_{N-1}$
Digit extract
$\tilde{m}_0$ ··· $\tilde{m}_{n-1}$
···
$\tilde{m}_{N-n}$ ··· $\tilde{m}_{N-1}$
Lin. transform
$m = \sum m_i X^i$

Fig. 3. The abstract bootstrapping procedure for BFV, with using slots. Each rectangle represents one ciphertext.

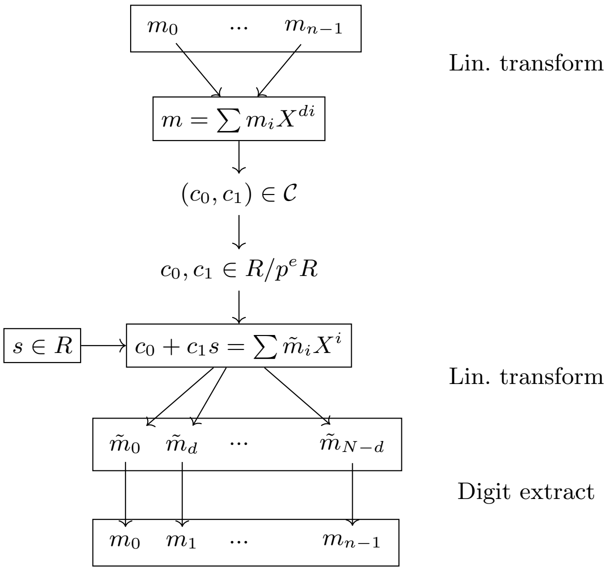
m_0 ··· m_{n-1}
Lin. transform
$m = \sum m_i X^{di}$
$(c_0, c_1) \in \mathcal{C}$
$c_0, c_1 \in R/p^e R$
$s \in R$
$c_0 + c_1 s = \sum \tilde{m}_i X^i$
Lin. transform
$\tilde{m}_0$ $\tilde{m}_d$ ··· $\tilde{m}_{N-d}$
Digit extract
m_0 m_1 ··· m_{n-1}

Fig. 4. The slim bootstrapping procedure for BFV [12]. For the inner product step, the values we want to retrieve have to be in the coefficients, so we move them there first. A rectangle represents one ciphertext.

and choose K as the fixed field $K = L^G$. The general theory of prime ideals in number field extensions (see e.g. [46, Prop. 9.3]) now yields that $[R/\mathfrak{B} : \mathcal{O}_K/\mathfrak{p}] = d$, and hence the reduction map $R \to S$ maps $\mathcal{O}_K$ to $\mathbb{Z}/p^e\mathbb{Z}$. In other words, we have the commutative diagram:

$$\begin{tikzcd}
\mathcal{O}_L \arrow[d, "\mathrm{mod}\ \mathfrak{B}^e"'] & \mathcal{O}_K \arrow[l, hook', "\text{inclusion}"'] \arrow[d, "\mathrm{mod}\ (\mathfrak{B}^e \cap K)"] \\
S & \mathbb{Z}/p^e\mathbb{Z} \arrow[l, hook', "\text{inclusion}"']
\end{tikzcd}$$

where $L \to S$ maps $\xi \mapsto \zeta$. This clearly implies that

$$\mathrm{MiPo}_K(\xi) \equiv \mathrm{MiPo}(\zeta) \mod p^e. \tag{1}$$

In a second step, note that

$$\mathrm{Gal}(L/\mathbb{Q}) \cong (\mathbb{Z}/2N\mathbb{Z})^* \cong \mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/\frac{N}{2}\mathbb{Z}.$$

Galois theory now shows that for each $2 \mid l \mid N$, there are exactly two intermediate fields $L/K'/\mathbb{Q}$ with $[L : K'] = l$. Observe now that both

$$K_1 = \mathbb{Q}[\xi^l] \quad \text{and} \quad K_2 = \mathbb{Q}[\mathrm{Re}(\xi^{l/2})]$$

are such fields (they are different, since K_1 is imaginary and K_2 is real), with

$$\mathrm{MiPo}_{K_1}(\xi) = T^l - \xi^l \quad \text{and} \quad \mathrm{MiPo}_{K_2}(\xi) = T^l - 2\mathrm{Re}(\xi^{l/2})T^{l/2} + 1.$$

The second equation holds, since $\xi^{l/2} + \xi^{-l/2} = 2\mathrm{Re}(\xi^{l/2})$ and so $\xi^l + 1 = 2\mathrm{Re}(\xi^{l/2})\xi^{l/2}$. The claim now follows from Eq. (1) with $l = d$. □