Bootstrapping for BGV and BFV Revisited

Robin Geelen and Frederik Vercauteren

imec-COSIC, ESAT, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. We unify the state-of-the-art bootstrapping algorithms for BGV and BFV in a single framework, and show that both schemes can be bootstrapped with identical complexity. This result corrects a claim by Chen and Han (Eurocrypt 2018) that BFV is more efficient to bootstrap than BGV. We also fix an error in their optimized procedure for power-of-two cyclotomics, which occurs for some parameter sets.
Our analysis is simpler, yet more general than earlier work, in that it simultaneously covers both BGV and BFV. Furthermore, we also design and implement a high-level open source software library for bootstrapping in the Magma Computer Algebra System. It is the first library to support both BGV and BFV bootstrapping in full generality, with all recent techniques (including the above fixes) and trade-offs.

Keywords: Fully homomorphic encryption · Bootstrapping · Brakerski-Gentry-Vaikuntanathan · Brakerski-Fan-Vercauteren · Recryption.

1 Introduction

Homomorphic encryption (HE) allows an external party to perform meaningful computations on encrypted data [39]. During that process, the external party learns nothing at all about the content of the data. This feature is useful in a multitude of applications, ranging from secure voting systems [12] and private set intersection [9] to privacy-preserving machine learning [5].

Homomorphic encryption schemes achieve their goal by evaluating Boolean or arithmetic circuits over ciphertexts, which has a corresponding effect on the plaintexts that they encrypt. All current schemes are noise-based, i.e., each ciphertext contains a "noise" or "error" term that grows as the computation progresses along the evaluated circuit. Ciphertexts only decrypt correctly if the noise is small enough such that it does not interfere with the plaintext. Therefore, when the noise reaches its upper threshold during circuit evaluation, we need a method to reduce it back to a lower level before continuing the actual computation. This idea was first introduced in the influential work of Gentry [20] under the name *bootstrapping* or *recryption*. It enables circuits of arbitrary complexity, possibly even unknown when the scheme's parameters are instantiated.

The idea of bootstrapping is to reduce noise by making a scheme decrypt itself homomorphically. If the complexity of the scheme's decryption circuit is low enough, then the noise term of the resulting ciphertext will be smaller than in the

original one. Schemes without a bootstrapping procedure are called somewhat homomorphic encryption (SHE) schemes. Schemes augmented with a bootstrapping procedure are referred to as fully homomorphic encryption (FHE) schemes. They can evaluate arbitrary circuits by chaining arithmetic operations, and inserting bootstrapping in appropriate places.

In this paper, we perform a comparative study of two closely related homomorphic schemes known as BGV and BFV. Both can evaluate arithmetic circuits (consisting of additions and multiplications) over a plaintext space modulo some prime power $p^r \geqslant 2$. These schemes are useful in applications that need exact arithmetic, such as private information retrieval [14] and oblivious RAM [13].

1.1 Related Work

Homomorphic encryption schemes are divided in multiple generations, depending on the type of bootstrapping operation they employ. First generation schemes include, among others, Gentry's original idea from 2009 [20]. This type of scheme is no longer used, because it has no advantage over newer constructions. Second generation schemes are characterized by a slow and complex bootstrapping procedure, but they gain a performance benefit from amortizing the recryption procedure over parallel SIMD computations and a large number of residual multiplicative levels. Examples are BGV [7] and BFV [6,16] upon which this work is based. Another well-known scheme is CKKS [10], which allows for approximate computation over the complex numbers. Finally, the branch of third generation schemes begins with the work of Gentry et al. [25] who construct a very simple homomorphic encryption scheme based on matrix multiplication. It was noticed later that their scheme enables a very fast and simple bootstrapping procedure, which led to the construction of FHEW [15] and TFHE [11].

BGV bootstrapping was proposed by Gentry et al. [22] and later improved by Alperin-Sheriff and Peikert [1]. Follow-up works focused on further optimizations and efficient implementation: the most relevant works to this paper are the Halevi/Shoup [30] implementation for BGV in `HElib`,[1] and the Chen/Han [8] implementation for BFV in SEAL.[2] Halevi and Shoup support a very practical and wide range of parameters, whereas Chen and Han have a more restricted parameter set. It is worth noting that both implementations follow the same blueprint, even though they support two different schemes. Finally, we note that much attention was spent recently on bootstrapping for the approximate CKKS scheme [3,31,34,35]. Most procedures follow again the same outline as BGV and BFV, but over the complex numbers instead of the integers modulo p^r.

Bootstrapping third generation schemes has lower circuit complexity, faster execution time and less noise growth than second generation schemes. Therefore, some studies have reported a hybrid approach where the idea is to bootstrap a second generation ciphertext using a third generation scheme [32,36]. Although this technique can lead to smaller FHE parameters, none of these studies have

[1] See https://github.com/homenc/HElib for bootstrapping in `HElib`.
[2] This implementation was never publicly released.

reported their execution time, and a rough estimate shows that all these methods are much slower than the state-of-the-art for practical parameters.

1.2 Contributions

We summarize our contributions as follows:

- We give an overview of the state-of-the-art bootstrapping methods for BGV and BFV. Many details about bootstrapping are currently scattered throughout the literature, and there is no single article that describes the interrelationship between the two schemes. Furthermore, our analysis unifies bootstrapping for BGV and BFV, and as a result, shows their equivalence in time complexity. This adjusts the erroneous claim of Chen and Han [8] who state that BFV has a more efficient bootstrapping procedure than BGV. We also correct a second error in the work of Chen and Han that relates to their optimized linear transformations for power-of-two cyclotomics. Finally, our comprehensive analysis allows us to draw well-formed conclusions about asymptotic time complexities and trade-offs in the implementation.
- We develop a high-level software library for BGV and BFV bootstrapping in the Magma Computational Algebra System.[3] Since both schemes follow an identical bootstrapping blueprint, we cover them via the same interface and need to implement the recryption procedure only once.

Although our implementation of bootstrapping is much slower than optimized FHE libraries such as `HElib`, it represents bootstrapping correctly and in full generality as a high-level instruction trace. In fact, our library was used in the DARPA DPRIVE program for hardware benchmarking [19, 41]. Specifically, we imported our implementation in a custom hardware simulator, in order to verify correctness and to measure performance.

1.3 Outline

We follow a top-down approach: Section 2 starts from the necessary background about BGV and BFV, considering them as somewhat homomorphic encryption schemes. Section 3 explains the recryption procedure from a very high level, encapsulating the precise operation of the involved building blocks. Afterwards, we gradually work out the two main building blocks: the linear transformations (Sections 4 and 5) and digit extraction (Section 6). Finally, we compare complexities and trade-offs, and we discuss implementation details.

2 Background

The following sections introduce the necessary definitions and notations for the somewhat homomorphic version of the BGV and BFV scheme. We also introduce the notion of SIMD operations over encrypted data.

[3] Available at https://github.com/KULeuven-COSIC/Bootstrapping_BGV_BFV.

2.1 Definitions and Notations

Cyclotomic polynomials. BGV and BFV are built around the ring learning with errors (R-LWE) problem – a commonly used hardness assumption in cryptography that relies on the theory of cyclotomic number fields. Let $m \geqslant 1$ be an integer, and let $\omega_m \in \mathbb{C}$ be a primitive mth root of unity. The polynomial

$$\Phi_m(x) = \prod_{j \in \mathbb{Z}_m^*} (x - \omega_m^j)$$

is called the mth *cyclotomic polynomial*. Its degree is equal to $n = \varphi(m)$, where $\varphi(\cdot)$ is Euler's totient function. Although cyclotomic polynomials have complex roots, it can be proven that they have integer coefficients only [43]. The R-LWE problem is then defined over the ring $\mathcal{R} = \mathbb{Z}[x]/(\Phi_m(x))$, which is a subring of the cyclotomic number field $\mathcal{K}_m = \mathbb{Q}[x]/(\Phi_m(x))$.

The set of all *automorphisms* of $\mathcal{K}_m$ that fix the base field $\mathbb{Q}$ forms a group under the function composition operator. It is called the Galois group of $\mathcal{K}_m/\mathbb{Q}$ and is denoted by $\mathrm{Gal}(\mathcal{K}_m/\mathbb{Q})$. Concretely, the automorphisms are given by $\tau_j \colon x \mapsto x^j$ for $j \in \mathbb{Z}_m^*$. In fact, the group $\mathrm{Gal}(\mathcal{K}_m/\mathbb{Q})$ is even isomorphic to $\mathbb{Z}_m^*$, where the group isomorphism is explicitly constructed as $j \mapsto \tau_j$. Sometimes, we use multiplicative subgroups of $\mathbb{Z}_m^*$: the subgroup generated by $g_1, \ldots, g_t \in \mathbb{Z}_m^*$ is denoted by $\langle g_1, \ldots, g_t \rangle$.

Notations. For an integer $N \geqslant 2$, we write the quotient ring of $\mathcal{R}$ modulo N as $\mathcal{R}_N = \mathcal{R}/N\mathcal{R}$. All elements of $\mathcal{R}$, $\mathcal{R}_N$ and $\mathcal{K}_m$ are shown in bold lower case letters (e.g., $\boldsymbol{a} \in \mathcal{R}$) or explicitly as polynomials (e.g., $a(x) \in \mathcal{R}$). Reduction modulo $\Phi_m(x)$ is implicit. The quotient ring $\mathcal{R}_N$ is seen as a subset of $\mathcal{R}$ with coefficients in $[-N/2, N/2) \cap \mathbb{Z}$. We denote the infinity norm on the coefficient vector of a ring element $\boldsymbol{a} \in \mathcal{R}$ as $||\boldsymbol{a}||_\infty$. Row vectors are written as $\mathsf{v} \in \mathcal{R}^{1 \times \ell}$.

For $\boldsymbol{a} \in \mathcal{K}_m$ and a positive integer $N \geqslant 2$, we denote the centered reduction modulo N by $[\boldsymbol{a}]_N$. More precisely, it is the unique element in $N\mathcal{R}+\boldsymbol{a}$ that has all coefficients in the set $[-N/2, N/2)$. Similarly, we denote coefficient-wise flooring, ceiling and rounding to the nearest integer by $\lfloor \boldsymbol{a} \rfloor$, $\lceil \boldsymbol{a} \rceil$ and $\lfloor \boldsymbol{a} \rceil$ respectively. The result is rounded upwards when the input is in $\mathbb{Z} + 1/2$.

2.2 Somewhat Homomorphic Encryption

This section describes the Brakerski-Gentry-Vaikuntanathan (BGV) and the Brakerski-Fan-Vercauteren (BFV) schemes. Importantly, both schemes have the same interface: they encrypt and operate on plaintexts $\boldsymbol{m} \in \mathcal{R}_{p^r}$ for some prime power $p^r \geqslant 2$, referred to as the *plaintext modulus*.[4] We focus on this interface, and where possible, make abstraction of the underlying implementation.

[4] In their most general version, BGV and BFV support any positive integer as plaintext modulus. However, bootstrapping is restricted to prime powers only.

Ciphertext format. It is not necessary to understand the details of the key generation and encryption procedures, so instead, we only explain the ciphertext format. We denote the secret decryption key for BGV and BFV by $\boldsymbol{s} \in \mathcal{R}$. A ciphertext is a vector $(\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_q^2$ that satisfies

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \boldsymbol{m} + p^r \boldsymbol{e} \pmod{q} \tag{1}$$

in the case of BGV. For obvious reasons, the parameter q is called the *ciphertext modulus*. For the BFV scheme, a valid ciphertext satisfies

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \lfloor (q/p^r) \cdot \boldsymbol{m} \rceil + \boldsymbol{e} \pmod{q}. \tag{2}$$

The term $\boldsymbol{e}$ is called the noise term, and just after encryption, it is guaranteed to have small coefficients with respect to q/p^r.

Decryption removes the noise term of the ciphertext in order to recover the message: we first compute the "inner product" with the secret key, which is identical for both schemes: $\boldsymbol{w} \leftarrow \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}$. Then we retrieve the plaintext as $\boldsymbol{m} \leftarrow [\boldsymbol{w}]_{p^r}$ for BGV and $\boldsymbol{m} \leftarrow [\lfloor (p^r/q) \cdot \boldsymbol{w} \rceil]_{p^r}$ for BFV. The result is correct if the coefficients of the noise term are less than $\lfloor q/(2p^r) \rfloor$.

The crucial difference between BGV and BFV is their plaintext encoding. As already suggested by Equations (1) and (2), the BGV scheme encodes the plaintext in the "least significant bits" of the ciphertext, whereas BFV uses the "most significant bits". In that sense, both schemes can be considered "dual" with only minor differences in their implementation of homomorphic multiplication [2].

Homomorphic operations. BGV and BFV support the exact same homomorphic operations, which can be summarized as follows:

- Addition takes two ciphertexts, and outputs a ciphertext that encrypts the sum of the underlying plaintexts. Similarly, we can also add a ciphertext and a plaintext together. Addition is a cheap operation, both in execution time and noise growth. Adding two ciphertexts increases the number of bits in the noise by at most one.
- Multiplication takes two ciphertexts, and outputs a ciphertext that encrypts the product of the underlying plaintexts. The output ciphertext is computed by tensoring the inputs, and is therefore a tuple of three elements. This can be reduced back to two elements by means of key switching (explained next). We can also multiply a ciphertext by a plaintext, and then the result is a standard tuple of just two elements. Multiplication is much more expensive than addition, especially in terms of noise growth. Multiplying two ciphertexts increases the number of bits in the noise by a ring-dependent parameter.
- The product of two ciphertexts consists of three ring elements, as opposed to only two elements in the standard format. To keep the ciphertext size under control, we need a post-processing step that converts the ciphertext back into the standard format. This is done by means of key switching. It is a very slow operation, but it adds (almost) no noise.

- The BGV scheme requires an extra operation to manage noise magnitude before each multiplication. This operation is called modulus switching, and it converts a ciphertext encrypted under modulus q to a different modulus q'. Modulus switching for the BFV scheme is not required for noise management, but it can still be applied to speed up the other homomorphic operations as working with a smaller modulus is more efficient.
- Given a plaintext modulus $p^{r_1+r_2}$ and an encryption of $p^{r_1}\boldsymbol{m}$, we perform exact division to compute an encryption of $\boldsymbol{m}$ under plaintext modulus p^{r_2}. In the BFV scheme, this operation comes for free; for BGV, the complexity is similar to multiplication by a constant. This operation does not add noise.

We emphasize that the minor differences between BGV and BFV do not have a large impact on performance. Recently, Kim et al. [33] studied the differences between BGV and BFV (without bootstrapping). They obtain similar results for both schemes, with only negligible differences in terms of noise growth (where BFV performs somewhat better) and computational complexity (where BGV performs somewhat better). Another result by Alperin-Sheriff and Peikert [1] shows that it is even possible to convert a ciphertext from BGV to BFV format or the other way around.

2.3 Plaintext Slots

Smart and Vercauteren [42] observed that one can encode multiple elements in a plaintext using the Chinese remainder theorem. The plaintext ring can then be seen as a vector of "plaintext slots", enabling homomorphic SIMD operations in a smaller ring. Concretely, if $\gcd(m, p) = 1$, then the polynomial $\Phi_m(x)$ splits over $\mathbb{Z}_{p^r}$ into ℓ distinct irreducible factors of degree d as

$$\Phi_m(x) = F_1(x) \cdot \ldots \cdot F_\ell(x) \pmod{p^r}. \tag{3}$$

The parameter d is the multiplicative order of p modulo m, and the number of factors is $\ell = n/d$, where $n = \varphi(m)$ is the degree of $\Phi_m(x)$.

Remember that the plaintext ring is $\mathcal{R}_{p^r} = \mathbb{Z}_{p^r}[x]/(\Phi_m(x))$. The Chinese remainder theorem allows us to write this ring as a direct sum of $\mathbb{Z}_{p^r}$-algebras through the isomorphism

$$\alpha(x) \mapsto (\alpha(x) \bmod F_1(x), \ldots, \alpha(x) \bmod F_\ell(x)) \tag{4}$$

that sends an element $\alpha(x)$ to its residue classes modulo each factor of $\Phi_m(x)$. The right-hand side of the isomorphism corresponds to a component-wise addition and multiplication, and the entries are called the *plaintext slots*.

It can be proven that the rings $\mathbb{Z}_{p^r}[x]/(F_i(x))$ are all isomorphic [22], and we can therefore talk about a unique *slot algebra*. That is, let ζ denote the residue class of x in the ring $\mathbb{Z}_{p^r}[x]/(F_1(x))$, then we define the slot algebra [28, 29] as $E = \mathbb{Z}_{p^r}[\zeta]$. It can be shown that Equation (4) simplifies to

$$\alpha(x) \mapsto \{\alpha(\zeta^h)\}_{h \in S}, \tag{5}$$

where $S \subseteq \mathbb{Z}$ is any complete system of representatives for $\mathbb{Z}_m^*/\langle p \rangle$. As such, the plaintext slots contain the values $\alpha(\zeta^h)$, where h ranges over S.

2.4 Hypercube Structure and Permutations

Gentry et al. [23] have shown that arbitrary permutations of the plaintext slots can be computed homomorphically. The permutations are based on the algebraic structure of the set S that was introduced in the previous section. This set can be constructed as

$$S = \{g_1^{e_1} \cdot \ldots \cdot g_t^{e_t} \mid 0 \leqslant e_i < \ell_i\}, \tag{6}$$

where the number of plaintext slots is equal to $|S| = \ell = \ell_1 \cdot \ldots \cdot \ell_t$.

Equation (6) induces a so-called *hypercube* structure on the plaintext slots, where t is the number of dimensions and ℓ_i is the size of dimension i. A visualization of a three-dimensional hypercube of size $3 \times 4 \times 3$ is given in Figure 1. Also one *hypercolumn* in the vertical dimension is highlighted. Each entry in the hypercube can be seen as storing one of the plaintext slots, indexed by a tuple of the form $(e_1, \ldots, e_t)$ with $0 \leqslant e_i < \ell_i$. Specifically, let $h = g_1^{e_1} \cdot \ldots \cdot g_t^{e_t}$, then we define the value of plaintext $\alpha(x)$ at the corresponding index to be $\alpha(\zeta^h)$.

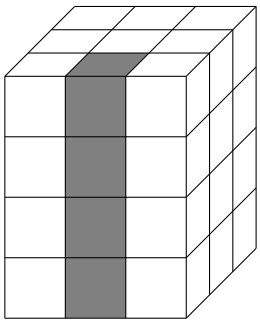


Fig. 1: A hypercube and a hypercolumn, adapted from [29]

A basic operation is permuting the plaintext slots circularly along one of the dimensions of the hypercube. These special permutations are called one-dimensional "rotations", and they are much more efficient to compute than arbitrary permutations. A rotation over $0 \leqslant v < \ell_i$ positions in dimension i is indicated by ρ_i^v. It maps the plaintext slot from index $(e_1, \ldots, e_i, \ldots, e_t)$ to index $(e_1, \ldots, e_i', \ldots, e_t)$ with $e_i' = e_i + v \pmod{\ell_i}$.

A rotation can be implemented by means of the automorphism group of $\mathcal{K}_m$. Let $\boldsymbol{\mu}$ be the "mask" obtained by embedding the constant '0' in the plaintext slots with indices $(e_1, \ldots, e_i, \ldots, e_t)$ where $e_i < v$, and embedding '1' in all other slots. Then for a plaintext $\boldsymbol{\alpha} \in \mathcal{R}_{p^r}$, the rotation ρ_i^v can be computed as

$$\rho_i^v(\boldsymbol{\alpha}) = \boldsymbol{\mu} \cdot \tau_j(\boldsymbol{\alpha}) + (1 - \boldsymbol{\mu}) \cdot \tau_k(\boldsymbol{\alpha}), \tag{7}$$

where $j = g_i^{-v} \pmod{m}$ and $k = g_i^{\ell_i - v} \pmod{m}$.

Sometimes, these permutations can even be implemented using just one automorphism instead of two. If the order of g_i in $\mathbb{Z}_m^*$ is equal to ℓ_i, then Equation (7) collapses to $\rho_i^v(\boldsymbol{\alpha}) = \tau_j(\boldsymbol{\alpha})$. Such a dimension is called "good"; if this is not the case, the dimension is called "bad".

Just like addition and multiplication, automorphisms can be computed over the ciphertext space. However, this operation requires post-processing by means of key switching (similar to ciphertext multiplication), which makes the resulting procedure expensive.

3 Bootstrapping Procedure

We now explain the bootstrapping operation – a general transformation from a somewhat into a fully homomorphic encryption scheme. This idea was proposed by Gentry [20] and is the only known way to support circuits of unbounded complexity. His construction "refreshes" a ciphertext when the noise term reaches its maximum level. This is achieved via homomorphic evaluation of a scheme's own decryption circuit, relying on an encryption of the secret key under itself, called the *bootstrapping key*.

From the above explanation, it follows that a somewhat homomorphic scheme can be turned into a fully homomorphic scheme if it can evaluate its own decryption circuit plus at least one more homomorphic operation. This can be realized by bootstrapping a ciphertext just before the noise level gets too high to evaluate an addition or multiplication. The idea is schematically represented in Figure 2. Analogous to regular decryption in the top of the diagram, we evaluate decryption homomorphically in the bottom. This results in an encryption of the same plaintext, but with a smaller and fixed noise level depending on the complexity of decryption. We need to augment the untrusted party with an encryption of the secret key. This can be done without compromising security of the system, assuming circular security of the somewhat homomorphic scheme [20].

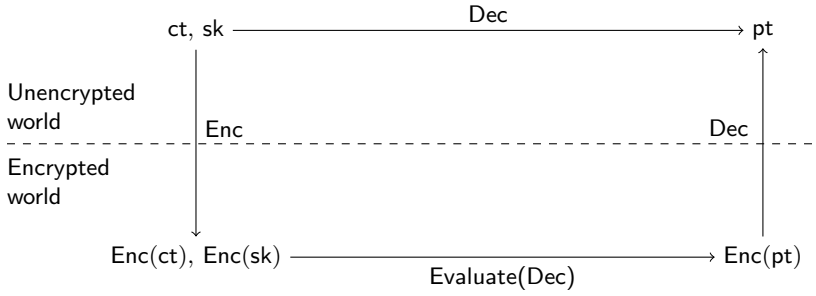


Fig. 2: Bootstrapping diagram

3.1 Bootstrapping for BGV and BFV

This section introduces the bootstrapping procedure for BGV and BFV from a very high level. We explain the functionality of the two main building blocks: the linear transformations and digit extraction. The details and implementation of these building blocks are deferred to Sections 4, 5 and 6.

Our derivations are simpler and more general than earlier work: firstly, we cover BGV and BFV in one analysis, emphasizing the similarities between both schemes. This is reflected by our implementation that differentiates between BGV and BFV in just one function call (the inner product); secondly, we start from a simplified version of decryption that does not require the "make-divisible" operation from `HElib`. This greatly simplifies the noise analysis, and moreover, we show that the same theory can be reused for the BFV scheme.

3.2 Simplifying the Decryption Function

Bootstrapping evaluates decryption homomorphically, so we start by pruning the decryption circuit as much as possible. In essence, we rewrite decryption such that it can be evaluated as cheaply as possible in the homomorphic domain. This is necessary for bootstrapping only, and is not relevant for normal decryption. Both for BGV and BFV, decryption is rewritten with respect to an additional parameter $e > r$ that determines the complexity of the resulting procedure; both the multiplicative depth of bootstrapping and the number of ciphertext operations will depend heavily on the magnitude of e. Additional information about the choice of e in a practical setting is provided later.

The BGV scheme. We start from a ciphertext encrypted with respect to a modulus $q = 1 \pmod{p^e}$, which can be obtained via modulus switching.[5] As shown in Section 3.3, the BGV ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1)$ can then be decrypted as

$$\boldsymbol{c}'_i \leftarrow [p^{e-r}\boldsymbol{c}_i]_q, \quad \boldsymbol{w} \leftarrow [\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}]_{p^e} \quad \text{and} \quad \boldsymbol{m} \leftarrow [\lfloor \boldsymbol{w}/p^{e-r} \rceil]_{p^r}. \tag{8}$$

The first step can be interpreted as scaling up the message with an additional factor of p^{e-r}. The second step is the usual "inner product" of decryption, and the last step removes the noise.

The BFV scheme. The simplified decryption function follows a similar outline as BGV. If we start from a ciphertext with respect to a modulus q, then the BFV ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1)$ can be decrypted via a standard scale-and-round procedure. More specifically, we compute

$$\boldsymbol{c}'_i \leftarrow \left[\left\lfloor \frac{p^e}{q}\boldsymbol{c}_i \right\rceil\right]_{p^e}, \quad \boldsymbol{w} \leftarrow [\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}]_{p^e} \quad \text{and} \quad \boldsymbol{m} \leftarrow [\lfloor \boldsymbol{w}/p^{e-r} \rceil]_{p^r}. \tag{9}$$

[5] This requirement can be relaxed to $\gcd(q, p) = 1$ if we multiply the second part of Equation (8) by q^{-1} and the third part by q, where $q^{-1} \cdot q = 1 \pmod{p^e}$. However, we only treat $q = 1 \pmod{p^e}$ so that Equation (8) is nearly identical for BFV.

The first step can be interpreted as modulus switching from q to p^e. Interestingly, the last two steps are identical to the BGV case.

The duality of BGV and BFV is again highlighted by their very similar decryption procedure. The only difference is in the first step, which can be interpreted as either scaling up the message from Equation (1), or scaling down the message from Equation (2). As we will see later, the main computational bottleneck of bootstrapping is not in the first step of decryption. This will result in identical computational complexities for both schemes, when expressed as the number of primitive homomorphic operations (i.e., when ignoring the differences in complexity resulting from the implementation).

3.3 Determining the Decryption Bound

The performance of bootstrapping depends heavily on the choice of e. If we take it very large, then performance will deteriorate; however, if we take it very small, then bootstrapping will fail. The purpose of this section is to find an appropriate value for e in practice, which can be seen as a trade-off between performance on the one hand, and failure probability on the other hand.

The BGV scheme. We show correctness of the simplified decryption procedure and at the same time derive equations for the noise. Let $\boldsymbol{u} = \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}$, then it follows that

$$\boldsymbol{u} = p^{e-r}(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) = p^{e-r}(\boldsymbol{m} + p^r \boldsymbol{e}) \pmod{q}.$$

Making the reduction modulo q explicit, and following the simplified decryption procedure, we have

$$\boldsymbol{w} = [\boldsymbol{u}]_{p^e} = [p^{e-r}\boldsymbol{m} + \boldsymbol{r}]_{p^e} \quad \text{for} \quad \boldsymbol{u} = p^{e-r}(\boldsymbol{m} + p^r \boldsymbol{e}) + q\boldsymbol{r}, \tag{10}$$

where we have used $q = 1 \pmod{p^e}$. Again, we follow the simplified decryption procedure and get

$$[\lfloor \boldsymbol{w}/p^{e-r} \rceil]_{p^r} = [\boldsymbol{m} + \lfloor \boldsymbol{r}/p^{e-r} \rceil]_{p^r} = \boldsymbol{m}$$

where the last equation is correct if the coefficients of $\boldsymbol{r}$ are upper bounded as $||\boldsymbol{r}||_\infty < p^{e-r}/2$. Applying the triangle inequality on Equation (10), we have

$$||\boldsymbol{r}||_\infty \leqslant ||(\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s})/q||_\infty + ||p^{e-r}\boldsymbol{m}/q||_\infty + ||p^e \boldsymbol{e}/q||_\infty < p^{e-r}/2.$$

For practical parameter settings, the second term is negligible compared to the third term and can be made arbitrarily small compared to the first term; we can therefore simply ignore it. Letting $\boldsymbol{d}_i = \boldsymbol{c}'_i/q$, the equation simplifies to

$$||\boldsymbol{d}_0 + \boldsymbol{d}_1 \cdot \boldsymbol{s}||_\infty + ||p^e \boldsymbol{e}/q||_\infty < p^{e-r}/2. \tag{11}$$

The BFV scheme. Consider the rounding error $\boldsymbol{d}_i = \boldsymbol{c}'_i - (p^e/q) \cdot \boldsymbol{c}_i$. Filling it in Equation (9), we get

$$\begin{aligned} \boldsymbol{w} = \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s} &= \frac{p^e}{q}\left(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}\right) + \left(\boldsymbol{d}_0 + \boldsymbol{d}_1 \cdot \boldsymbol{s}\right) \\ &= \frac{p^e}{q}\left(\frac{q}{p^r}\boldsymbol{m} + \boldsymbol{e}\right) + \left(\boldsymbol{d}_0 + \boldsymbol{d}_1 \cdot \boldsymbol{s}\right) \\ &= p^{e-r}\boldsymbol{m} + \left(\boldsymbol{d}_0 + \boldsymbol{d}_1 \cdot \boldsymbol{s}\right) + p^e \boldsymbol{e}/q \qquad (\text{mod } p^e). \end{aligned}$$

Decryption works properly if the second and third term are small enough so that they can be removed during the rounding procedure. Formally, we require

$$||\boldsymbol{d}_0 + \boldsymbol{d}_1 \cdot \boldsymbol{s}||_\infty + ||p^e \boldsymbol{e}/q||_\infty < p^{e-r}/2. \tag{12}$$

An important observation is that Equations (11) and (12) are identical, so we can cover BGV and BFV in one analysis. The equations consist of two separate terms. The first term depends on $\boldsymbol{d}_i$ and $\boldsymbol{s}$. Due to the seemingly random properties of ciphertexts, one can make the heuristic assumption that the coefficients of $\boldsymbol{d}_i$ are uniformly distributed in the interval $[-1/2, 1/2)$ and independent of $\boldsymbol{s}$. The size of the first term can then be analyzed based on statistical properties of the secret key. The second term represents the noise and is ciphertext-specific.

Statistical analysis. Halevi and Shoup [30] carry out a heuristic analysis on Equations (11) and (12). They start from a secret key with coefficients in $\{0, \pm 1\}$ and sampled uniformly with Hamming weight equal to a parameter denoted h. The end result is the probability of a bootstrapping failure. The analysis is quite extensive, so we will only state the results and give some intuition.

The dominant term in the decryption bound is $\boldsymbol{d}_1 \cdot \boldsymbol{s}$. Therefore, we compute the probability that this term exceeds a certain threshold that is parameterized by a real number k. Specifically, Halevi and Shoup show that

$$\Pr\left[||\boldsymbol{d}_1 \cdot \boldsymbol{s}||_\infty > k \cdot C\right] < D \cdot \operatorname{erfc}(k/\sqrt{2}), \tag{13}$$

where erfc is the complementary error function. The parameters are given by

$$C = \frac{1}{\sqrt{12}} \cdot 2^{t/2} \cdot \sqrt{h} \cdot \sqrt{\frac{\varphi(m)}{m}} \qquad \text{and} \qquad D = \varphi(m),$$

where m is the cyclotomic index and t is its number of distinct prime factors.[6] We conclude that

- The probability of Equation (13) grows proportionally to the lattice dimension $n = \varphi(m)$ because of the factor D.

[6] In fact, the analysis only holds if the linear transformations from Section 4 exploit the prime-power factorization of m.

- For constant m and k (which represents constant failure probability), C is directly proportional to $\sqrt{h}$. Neglecting $\boldsymbol{d}_0$ and $\boldsymbol{e}$ in Equations (11) and (12), we need to take $p^{e-r}/2$ proportional to $\sqrt{h}$. The parameter r is usually fixed by the application, so we take p^e proportional to $\sqrt{h}$.
- Choosing a concrete value of e is done by first picking k, depending on the desired failure probability. Then we compute the smallest e such that the decryption bound of Equations (11) and (12) is satisfied.

Recall that the complexity of bootstrapping depends on e, which is in term directly proportional to $\sqrt{h}$. In fact, the situation is similar in the CKKS scheme for which Bossuat et al. [4] circumvent the problem using a clever trick: they evaluate CKKS bootstrapping by decrypting homomorphically under a different secret key $\tilde{\boldsymbol{s}}$ of Hamming weight $\tilde{h} < h$. This is done by first obtaining an encryption of the message under $\tilde{\boldsymbol{s}}$ via key switching. Despite the more efficient attacks on sparse secrets, the proposed method does not hurt security: since key switching is done at a sufficiently small modulus, we can compensate for the security loss caused by a sparser secret.

The techniques from Bossuat et al. carry over to BGV and BFV directly: before homomorphic decryption, we can switch the input ciphertext to a sparse key $\tilde{\boldsymbol{s}}$. Then the second subequation of Equations (8) and (9) is evaluated homomorphically using $\tilde{\boldsymbol{s}}$ instead of $\boldsymbol{s}$. This requires a slightly different bootstrapping key, namely an encryption of $\tilde{\boldsymbol{s}}$ under $\boldsymbol{s}$.

3.4 High-Level Overview of the Bootstrapping Procedure

We are now ready to describe the bootstrapping procedure from a high level. We differentiate general bootstrapping where the plaintext slots contain elements from E, and thin bootstrapping where the plaintext slots are restricted to $\mathbb{Z}_{p^r}$.

General bootstrapping. The general bootstrapping procedure starts from a ciphertext with fully packed slots. Bootstrapping consists of the following steps, which are also shown in Figure 3:

- Inner product: evaluate the first and second subequation of Equation (8) (for BGV) or Equation (9) (for BFV). The second subequation is evaluated homomorphically, where we process the bootstrapping key under plaintext modulus p^e. This step is the only one that is different for BGV and BFV. There also exists a variant of the inner product step which does not require a bootstrapping key [1], but it is not discussed here.
- Linear transformation: move the noisy coefficients of the encrypted plaintext into the slots, encoding one coefficient per slot. Since there are d times more coefficients than slots, we end up with d ciphertexts instead of one.
- Digit extraction: perform a slot-wise rounding procedure on each of the d ciphertexts. This corresponds to the last subequation of Equation (8) or (9), and decreases the plaintext modulus from p^e to p^r.

– Inverse linear transformation: move the noise-free slots of the encrypted plaintexts back into the coefficients.

Note that the linear transformations are necessary to do the rounding element-wise. Digit extraction cannot be done directly on the coefficients as it involves ciphertext multiplication, which is not coefficient-wise.

$$
\begin{array}{rl}
\mathsf{Enc}(\boldsymbol{m}) & \\
\downarrow & \text{Inner product} \\
\mathsf{Enc}(p^v \cdot \boldsymbol{m} + \boldsymbol{e}) & \\
\downarrow & \text{Linear transformation} \\
\mathsf{Enc}(p^v \cdot m_{\ell i} + e_{\ell i}, \ldots, p^v \cdot m_{\ell(i+1)-1} + e_{\ell(i+1)-1}) & \text{for } i = 0, \ldots, d-1 \\
\downarrow & \text{Digit extractions} \\
\mathsf{Enc}(m_{\ell i}, \ldots, m_{\ell(i+1)-1}) & \text{for } i = 0, \ldots, d-1 \\
\downarrow & \text{Inverse linear transformation} \\
\mathsf{Enc}(\boldsymbol{m}) &
\end{array}
$$

Fig. 3: General bootstrapping procedure, adapted from [8]

Thin bootstrapping. The thin bootstrapping procedure starts from a ciphertext with sparsely packed slots. Bootstrapping consists of the following steps, which are also shown in Figure 4:

– Linear transformation: move the noise-free slots of the encrypted plaintext into the coefficients.
– Inner product: evaluate the first and second subequation of Equation (8) (for BGV) or Equation (9) (for BFV). This step is identical to the corresponding step from the general bootstrapping procedure.
– Inverse linear transformation: move the noisy coefficients of the encrypted plaintext back into the slots, encoding one coefficient per slot.
– Digit extraction: perform a slot-wise rounding procedure on the ciphertext. This step is identical to the corresponding step from the general bootstrapping procedure, except that we only have one ciphertext instead of d.

The main difference with the general bootstrapping procedure is that we exploit the sparsely packed slots by changing the order of operations: by moving the slots directly into the coefficients, the number of ciphertexts stays just one. Therefore, we also require just one digit extraction instead of d.

4 Homomorphic Linear Transformations

Linear transformations are an important aspect of recryption, and essential for mapping coefficients to slots and back. We will need E-linear and $\mathbb{Z}_{p^r}$-linear

$$\begin{array}{c}
\mathsf{Enc}(m_0, \ldots, m_{\ell-1}) \\
\downarrow \text{Linear transformation} \\
\mathsf{Enc}(\boldsymbol{m}) \\
\downarrow \text{Inner product} \\
\mathsf{Enc}(p^v \cdot \boldsymbol{m} + \boldsymbol{e}) \\
\downarrow \text{Inverse linear transformation} \\
\mathsf{Enc}(p^v \cdot m_0 + e_0, \ldots, p^v \cdot m_{\ell-1} + e_{\ell-1}) \\
\downarrow \text{Digit extraction} \\
\mathsf{Enc}(m_0, \ldots, m_{\ell-1})
\end{array}$$

Fig. 4: Thin bootstrapping procedure, adapted from [8]

transformations: the notion of E-linearity is defined with respect to the right-hand side of Equation (5), which is a module over E; $\mathbb{Z}_{p^r}$-linearity follows from the ring structure of $\mathcal{R}_{p^r}$, which is a module over $\mathbb{Z}_{p^r}$.

4.1 One-Dimensional Linear Transformations

We restrict ourselves to the case of one-dimensional linear transformations, i.e., transformations that act on the hypercolumns in a certain dimension separately. Figure 5 shows a schematic representation of a one-dimensional linear transformation. The $2 \times 4 \times 2$ hypercube is reinterpreted as a series of hypercolumns in the vertical dimension. Each hypercolumn is then interpreted as a vector (either an E-vector or a $\mathbb{Z}_{p^r}$-vector), and a matrix multiplication is done for each of the vectors separately. For example, the input-output relation for an E-linear transformation of the front right hypercolumn is given by the matrix-vector product

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} = M \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \end{bmatrix}$$

with $M \in E^{4\times 4}$. The same is done for all other hypercolumns in the vertical dimension, where the matrix M may be different for each hypercolumn.

***E*-linear transformations.** A one-dimensional E-linear transformation can be computed by multiplying each hypercolumn with a matrix $M \in E^{\ell_i \times \ell_i}$. For a vector $\alpha \in E^{\ell_i}$, let α_v be the vector obtained by rotating the entries of α by v positions. Using some algebraic manipulations, it was established by Halevi and Shoup [26] that we can rewrite the matrix-vector product as

$$M\alpha = \sum_{v=0}^{\ell_i - 1} M_v \alpha_v \tag{14}$$

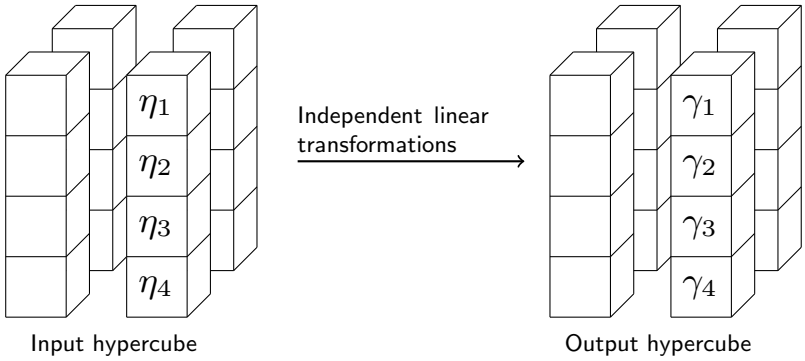


Fig. 5: A one-dimensional linear transformation

where each M_v is a diagonal matrix.

Now instead of a vector, consider an entire plaintext $\boldsymbol{\alpha} \in \mathcal{R}_{p^r}$. Using the above theory, any one-dimensional E-linear transformation can be written as

$$L(\boldsymbol{\alpha}) = \sum_{v=0}^{\ell_i - 1} \boldsymbol{\kappa}_v \cdot \rho_i^v(\boldsymbol{\alpha}), \tag{15}$$

where the constants $\boldsymbol{\kappa}_v$ are obtained by embedding the diagonal entries of M_v in the slots. A naive approach can implement Equation (15) with ℓ_i constant-ciphertext multiplications and $\ell_i - 1$ rotations. However, Halevi and Shoup [28] have proposed a baby-step/giant-step algorithm that is cheaper. Their idea is to write Equation (15) as a double summation, and as such, it requires $\mathcal{O}(\sqrt{\ell_i})$ automorphisms only. Their algorithm is slightly more expensive in bad hypercube dimension than in good dimensions. Since linear transformations are a separate research direction that can be decoupled from bootstrapping, we omit further details. We refer to `HElib` for implementation aspects [28].

$\mathbb{Z}_{p^r}$-linear transformations. A one-dimensional $\mathbb{Z}_{p^r}$-linear transformation can be computed using roughly the same strategy as for an E-linear one. We require one special automorphism, called the *Frobenius map* $\sigma = \tau_p$. It acts on each plaintext slot individually as the slot-wise Frobenius map $\sigma_E \colon \beta(\zeta) \mapsto \beta(\zeta^p)$. The slot-wise Frobenius map is a $\mathbb{Z}_{p^r}$-linear function from E to E. Conversely, any $\mathbb{Z}_{p^r}$-linear function $L \colon E \to E$ can be written as a linear combination of Frobenius powers, i.e.,

$$L(\eta) = \sum_{f=0}^{d-1} \theta_f \cdot \sigma_E^f(\eta). \tag{16}$$

The constants $\theta_f \in E$ can be obtained with field theory for the prime case [40] and can then be Hensel lifted to the prime-power case [30]. Furthermore, when the image of L lies in $\mathbb{Z}_{p^r}$, the constants are related as $\theta_f = \sigma_E^f(\theta_0)$.

Equation (14) remains valid for $\mathbb{Z}_{p^r}$-linear transformations, but now the matrix entries of M are themselves $\mathbb{Z}_{p^r}$-linear maps from E to E. Any one-dimensional $\mathbb{Z}_{p^r}$-linear transformation can therefore be written as

$$L(\boldsymbol{\alpha}) = \sum_{v=0}^{\ell_i - 1} \sum_{f=0}^{d-1} \boldsymbol{\kappa}_{v,f} \cdot \sigma^f(\rho_i^v(\boldsymbol{\alpha})), \tag{17}$$

where the constants $\boldsymbol{\kappa}_{v,f}$ are obtained by embedding appropriate values in the slots. Those values can be found by considering the diagonal entries of M_v and solving a system of linear equations to determine the θ_f's from Equation (16). A naive approach can implement Equation (17) with $d \cdot \ell_i$ constant-ciphertext multiplications, $\ell_i - 1$ rotations and $(d-1) \cdot \ell_i$ Frobenius powers. However, Halevi and Shoup [28] have proposed a baby-step/giant-step algorithm that is cheaper and requires $\mathcal{O}(d + \ell_i)$ automorphisms only.

5 The Slot-to-Coefficient Transformation

In this section, we discuss two different methods for mapping plaintext slots to coefficients and back. This is an essential part of the bootstrapping procedure from Section 5. Both methods are based on homomorphic linear transformations. The first one was introduced by Halevi and Shoup [30] and supports a broad range of parameters. The second one was introduced by Chen and Han [8], but is more restrictive since it can only be used for power-of-two cyclotomics. However, the second method can slightly outperform the first one for some parameter sets.

5.1 The General Case

We now explain the *evaluation map*, which is a linear transformation designed by Halevi and Shoup [27,30] for `HElib`. The evaluation map moves the plaintext slots of some element $\beta(x)$ into the coefficients of some element $\alpha(x)$. That is, if the slots of $\beta(x)$ encode $c_1, \ldots, c_n$, then the output of the evaluation map is the element $\alpha(x)$, the coefficients of which are precisely these numbers. We start with some algebraic background.

The powerful basis. Throughout this section, consider a pairwise coprime factorization of the cyclotomic index $m = m_1 \cdot \ldots \cdot m_t$. The cyclotomic ring $\mathcal{R}$ is then isomorphic to

$$\mathbb{Z}[x]/(\Phi_m(x)) \cong \mathbb{Z}[x_1, \ldots, x_t]/(\Phi_{m_1}(x_1), \ldots, \Phi_{m_t}(x_t)),$$

where the ring isomorphism is given by $x_i \mapsto x^{m/m_i}$. The representation in $\mathbb{Z}[x]/(\Phi_m(x))$ is done in the *power basis* consisting of $x^0, x^1, \ldots, x^{\varphi(m)-1}$. On the other hand, the representation in $\mathbb{Z}[x_1, \ldots, x_t]/(\Phi_{m_1}(x_1), \ldots, \Phi_{m_t}(x_t))$ leads to the *powerful basis* [37] consisting of $x_1^{j_1} \cdot \ldots \cdot x_t^{j_t}$ with $0 \leqslant j_i < \varphi(m_i)$.

Building the hypercube structure. Recall that the linear transformations rely on the representative set S for $\mathbb{Z}_m^*/\langle p \rangle$. In order to compute the evaluation map efficiently, it is necessary that S has a special structure corresponding to the factorization $m = m_1 \cdot \ldots \cdot m_t$. Consider the following lemma, where we define $\mathrm{CRT}(h_1, \ldots, h_t)$ to be the unique $h \in \{0, \ldots, m-1\}$ such that $h = h_i \pmod{m_i}$.

Lemma 1 ([30, Lemma 4.1]). *Consider the integers p and $m = m_1 \cdot \ldots \cdot m_t$ as before. Let d_i be the order of $p^{d_1 \cdot \ldots \cdot d_{i-1}}$ modulo m_i. Then the order of p modulo m is $d = d_1 \cdot \ldots \cdot d_t$, which is equal to the degree of the factors in Equation (3).*

Moreover, suppose that $S_1, \ldots, S_t \subseteq \mathbb{Z}$ are such that each S_i forms a complete system of representatives for $\mathbb{Z}_{m_i}^*/\langle p^{d_1 \cdot \ldots \cdot d_{i-1}} \rangle$. Then the set $S = \mathrm{CRT}(S_1, \ldots, S_t)$ forms a complete system of representatives for $\mathbb{Z}_m^*/\langle p \rangle$.

In order to simplify the evaluation map, some additional constraints are placed on the choice of the sets S_i:

- Each group $\mathbb{Z}_{m_i}^*/\langle p^{d_1 \cdot \ldots \cdot d_{i-1}} \rangle$ is supposed to be cyclic of order $\ell_i = \varphi(m_i)/d_i$ and with generator $\tilde{g}_i$.[7] The set S can then be built using Equation (6) by taking $g_i = \mathrm{CRT}(1, \ldots, 1, \tilde{g}_i, 1, \ldots, 1)$. This first restriction allows us to decompose the evaluation map as a series of one-dimensional linear transformations, namely one for each factor of m.
- We limit ourselves to the case $d_1 = d$ and $d_i = 1$ for $i = 2, \ldots, t$. This second restriction results in a $\mathbb{Z}_{p^r}$-linear transformation in the first dimension and an E-linear transformation in the other dimensions.[8]

Note that with these assumptions, the first dimension of the hypercube can be good or bad, but the other dimensions are always good.

In the rest of this section, we denote the powerful basis representation of some element $\alpha(x)$ by $\alpha'(x_1, \ldots, x_t)$. For $i = 1, \ldots, t$, we define

$$\zeta_i = \zeta^{m/m_i} \qquad \text{and} \qquad \zeta_{i,e} = \zeta_i^{g_i^e}.$$

It follows from the powerful basis representation that for $h = \mathrm{CRT}(h_1, \ldots, h_t)$ with $h_i \in \mathbb{Z}$, we have that $\alpha(\zeta^h) = \alpha'(\zeta_1^{h_1}, \ldots, \zeta_t^{h_t})$. Recall that the plaintext slots of $\alpha(x)$ contain the values $\alpha(\zeta^h)$, where h ranges over S. Equivalently, using Lemma 1, the plaintext slots hold $\alpha'(\zeta_1^{h_1}, \ldots, \zeta_t^{h_t})$, where the h_i's range over S_i, or even $\alpha'(\zeta_{1,e_1}, \ldots, \zeta_{t,e_t})$, where the e_i's range over $\{0, \ldots, \ell_i - 1\}$.

The evaluation map. The functionality of the evaluation map is moving slots to powerful basis coefficients. Halevi and Shoup [30] showed that it can be decomposed as a series of one-dimensional linear transformations. Suppose that we

[7] The non-cyclic case could be handled by multi-dimensional linear transformations. However, if we exploit the prime-power factorization of m, then this situation can only occur if m is divisible by 8 [17].

[8] Alleviating this restriction would result in intermediate F-linear transformations, where $\mathbb{Z}_{p^r} \subseteq F \subseteq E$. All stages of the evaluation map (explained later) can be treated as a special case of this situation.

start with the plaintext $\beta(x)$ and end with the plaintext $\alpha(x)$. Let the powerful basis coefficients of $\alpha(x)$ be given by $c_{j_1,\ldots,j_t}$, i.e.,

$$\alpha'(x_1,\ldots,x_t) = \sum_{j_1,\ldots,j_t} c_{j_1,\ldots,j_t}\, x_1^{j_1} \cdot \ldots \cdot x_t^{j_t},$$

then the plaintext slots of $\beta(x)$ encode these coefficients in some way. Specifically, consider a normal element θ from $E/\mathbb{Z}_{p^r}$, then the slot at index $(e_1,\ldots,e_t)$ holds

$$\beta'(\zeta_{1,e_1},\ldots,\zeta_{t,e_t}) = \sum_{f=0}^{d-1} c_{e_1+\ell_1\cdot f,e_2,\ldots,e_t}\, \sigma_E^f(\theta).$$

We show next that the evaluation map can be decomposed into t stages of one-dimensional linear transformations, where the output of stage i is denoted $\beta_i(x)$. The map is such that the slot of $\beta_i(x)$ at index $(e_1,\ldots,e_t)$ holds

$$\beta_i'(\zeta_{1,e_1},\ldots,\zeta_{t,e_t}) = \sum_{j_1,\ldots,j_i} c_{j_1,\ldots,j_i,e_{i+1},\ldots,e_t}\, \zeta_{1,e_1}^{j_1} \cdot \ldots \cdot \zeta_{i,e_i}^{j_i}.$$

Stage 1. The first stage takes the plaintext $\beta(x)$ as input and transforms it into $\beta_1(x)$. Consider the slot of $\beta_1(x)$ at index $(e_1,\ldots,e_t)$, and observe that it is a function of the slots of $\beta(x)$ at indices $(e_1',e_2,\ldots,e_t)$ with $0 \leqslant e_1' < \ell_1$. In other words, the output slot only depends on the input slots of the corresponding hypercolumn in dimension 1, so the transformation is one-dimensional. It is easy to see that the transformation is also $\mathbb{Z}_{p^r}$-linear, and therefore it can be implemented using the theory of Section 4.1. Specifically, we need to evaluate Equation (14) where the element of M at row $j+1$ and column $k+1$ is

$$m_{j+1,k+1}\colon \sum_{f=0}^{d-1} c_f\, \sigma_E^f(\theta) \mapsto \sum_{f=0}^{d-1} c_f\, \zeta_{1,j}^{k+\ell_1\cdot f} \tag{18}$$

for $c_f \in \mathbb{Z}_{p^r}$. The matrix is identical for each hypercolumn.

Stages 2, …, t. The ith stage takes the plaintext $\beta_{i-1}(x)$, i.e., the output from the previous stage, and transforms it into $\beta_i(x)$. Consider the slot of $\beta_i(x)$ at index $(e_1,\ldots,e_t)$, and observe that it is a function of the slots of $\beta_{i-1}(x)$ at indices $(e_1,\ldots,e_{i-1},e_i',e_{i+1},\ldots,e_t)$ with $0 \leqslant e_i' < \ell_i$. In other words, the output slot only depends on the input slots of the corresponding hypercolumn in dimension i, so the transformation is one-dimensional. It is easy to see that the transformation is also E-linear, and therefore it can be implemented using the theory of Section 4.1. Specifically, we need to evaluate Equation (14) where the element of M at row $j+1$ and column $k+1$ is $m_{j+1,k+1} = \zeta_{i,j}^k$. The matrix is identical for each hypercolumn.

The correctness of the above procedure follows from the fact that $\beta_t(x) = \alpha(x)$. This can be proven by observing that $\beta_t(x)$ and $\alpha(x)$ contain the same value in each plaintext slot. Finally, we note that the evaluation map can be inverted by running all stages in reverse order and with matrix M^{-1} instead of M.

Fully packed slots. In the following paragraphs, we explain how the evaluation map is deployed in the general bootstrapping procedure. Recall that the evaluation map starts from a ciphertext, the slots of which encode the numbers that we are interested in. However, digit extraction operates on sparsely packed ciphertexts that only encode one element in the constant term of each slot. We therefore need a *packing* procedure that converts d ciphertexts (all of them sparsely packed), into one fully packed ciphertext. Similarly, we need an *unpacking* procedure for the inverse evaluation map.

Unpacking. We start from an encryption of a plaintext $\boldsymbol{\beta}$ that contains d numbers per slot, encoded using a normal element $\theta \in E$. Unpacking can be computed based on the Frobenius map. Consider the constants $\boldsymbol{\kappa}_f$ corresponding to the linear map that acts on each slot as

$$L_0 \colon \sum_{f=0}^{d-1} c_f \, \sigma_E^f(\theta) \mapsto c_0 \tag{19}$$

for $c_f \in \mathbb{Z}_{p^e}$.[9] We first precompute $\sigma^f(\boldsymbol{\beta})$ for $f = 0, \ldots, d-1$. It can easily be verified that the ciphertexts

$$\boldsymbol{\beta}_i = \sum_{f=0}^{d-1} \boldsymbol{\kappa}_{f+i} \cdot \sigma^f(\boldsymbol{\beta})$$

contain only one of desired numbers per slot. Note that the index is implicitly reduced modulo d, so $\boldsymbol{\kappa}_{f+d} = \boldsymbol{\kappa}_f$ by definition.

Repacking. Conversely, we start from a set of plaintexts $\boldsymbol{\beta}_i$ that contain one number per slot. Repacking can be computed via the sum

$$\boldsymbol{\beta} = \sum_{f=0}^{d-1} \boldsymbol{\kappa}_f \cdot \boldsymbol{\beta}_f,$$

where $\boldsymbol{\kappa}_0$ contains θ in each slot and $\boldsymbol{\kappa}_f = \sigma^f(\boldsymbol{\kappa}_0)$.

Sparsely packed slots. Thin bootstrapping starts from a ciphertext with sparsely packed slots. This gives an opportunity for improving the linear maps, which we discuss separately for the forward and inverse transformation.

The forward map. The evaluation map can be simplified in this case: since the plaintext slots encode only a number in the constant term, we can alleviate the first stage. Indeed, the matrix entries of Equation (14) can be simplified to

$$m_{j+1,k+1} \colon \eta \mapsto \zeta_{1,j}^k \cdot \eta.$$

This corresponds to an E-linear transformation identical to stages $2, \ldots, t$.

[9] Recall that the inverse evaluation map is computed with respect to a higher precision plaintext space modulo p^e with $e > r$. Therefore the constants c_f come from $\mathbb{Z}_{p^e}$.

The inverse map. The analysis for the inverse evaluation map becomes slightly more complicated. The inner product step from Section 3.4 generates undesired coefficients that represent noise and do not correspond to the numbers we are interested in. Ignoring these extra coefficients would break the procedure, so we cannot just run the forward map in reverse order [28].

The solution to this problem is to run the inverse evaluation map with some adaptations, that is, stage 1 can be optimized. Assume that we start from an encryption of a plaintext $\boldsymbol{u}$. Since the coefficients of $\boldsymbol{u}$ are not sparse, we cannot directly implement stage 1 as an E-linear transformation. Instead, consider the matrix M with entries given by Equation (18), and consider the linear map L_0 from Equation (19). We must first apply M^{-1} to each hypercolumn and then perform a slot-wise unpacking with L_0. This is equivalent to a multiplication with LM^{-1}, where L is the diagonal matrix containing L_0 everywhere. The matrix LM^{-1} contains linear maps to the base ring, namely $\mathbb{Z}_{p^e}$ for both schemes. Hence each of its entries can be written as

$$\eta \mapsto \sum_{f=0}^{d-1} \sigma_E^f(\theta_0 \cdot \eta)$$

for some $\theta_0 \in E$. More specifically, denote by $\theta_{j,k}$ the constant corresponding to the jth row and the kth column of LM^{-1}. Moreover, let

$$T_E \colon \eta \mapsto \sum_{f=0}^{d-1} \sigma_E^f(\eta)$$

be the trace map on E. We can now rewrite the matrix as

$$LM^{-1} = \begin{bmatrix} T_E & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & T_E \end{bmatrix} \begin{bmatrix} \theta_{1,1} & \dots & \theta_{1,\ell_1} \\ \vdots & \ddots & \vdots \\ \theta_{\ell_1,1} & \dots & \theta_{\ell_1,\ell_1} \end{bmatrix},$$

so stage 1 can be decomposed as an E-linear transformation, followed by a slot-wise trace map. The slot-wise trace map can be efficiently implemented using the algorithm from `HElib` [26].

5.2 Power-of-Two Cyclotomics

Chen and Han propose an optimization for the evaluation map that is specific to power-of-two cyclotomics in conjunction with thin bootstrapping [8]. Their procedure is also more general in one aspect, namely it does not require the group $\mathbb{Z}_m^*/\langle p \rangle$ to be cyclic. However, their theoretical derivation contains an error that breaks the recryption procedure for some parameter sets. Specifically, they claim that the hypercube set S can be chosen in an arbitrary way. As we will show, this is not true for the coefficient-to-slot transformation.

Building the hypercube structure. In contrast to Chen and Han, we build the set S in a slightly more complex way. Specifically, we take

$$S = \{g_1^{e_1} \cdot g_2^{e_2} \mid 0 \leqslant e_i < \ell_i\},$$

where $\ell_1 = 2$ and $\ell_2 = \ell/2$. The generators g_1 and g_2 are computed using the following procedure:

1. Choose g_1 of order 2 in $\mathbb{Z}_m^*/\langle p \rangle$ such that the quotient group $\mathbb{Z}_m^*/\langle p, g_1 \rangle$ is cyclic. From all possible options, we take one such that $g_1 - 1$ has a 2-adic valuation equal to 1.
2. Choose g_2 as a generator of the quotient group $\mathbb{Z}_m^*/\langle p, g_1 \rangle$.

Choosing the set S in the above manner is always possible due to the special structure of the group of invertible integers modulo powers of two [17]. We show the correctness of this adapted procedure in Lemma 2.

The forward map. The slot-to-coefficient transformation maps a plaintext $\boldsymbol{\beta}$ of which the slots encode $(\alpha_0, \ldots, \alpha_{d(\ell-1)})$ to a plaintext $\boldsymbol{\alpha}$. The functionality of the transformation is defined as

$$\boldsymbol{\beta} \mapsto \boldsymbol{\alpha} = \sum_{j=0}^{\ell-1} \alpha_{dj} x^{dj}. \tag{20}$$

It can be expressed on the plaintext space as

$$\boldsymbol{\beta} \mapsto \sum_{h \in S} \boldsymbol{\kappa}_h \cdot \tau_h(\boldsymbol{\beta}), \tag{21}$$

where $\boldsymbol{\kappa}_h$ are precomputed constants. This can be seen as follows: since the slots are only sparsely packed, the automorphisms τ_h act directly as permutations over $\mathbb{Z}_{p^r}$-vectors. These permutations are arbitrary in the sense that they can map any slot to any other slot. Thus, each slot from the input can influence each slot from the output. Finally, we note that the constants $\boldsymbol{\kappa}_h$ can be computed by filling in Equations (20) and (21) for ℓ independent input-output pairs, and then solving the obtained system of linear equations for each slot separately.

The inverse map. The analysis for the inverse evaluation map becomes slightly more complicated. Similarly to the general case, the inner product step generates undesired coefficients that represent noise and do not correspond to the numbers we are interested in. Ignoring these extra coefficients would break the procedure, so we cannot just run the forward map in reverse order.

In order to remove the undesired coefficients, we run a *coefficient selection* procedure that is defined as

$$\sum_{j=0}^{n-1} \alpha_j x^j \mapsto \boldsymbol{\alpha} = \sum_{j=0}^{\ell-1} \alpha_{dj} x^{dj}.$$

In other words, we remove the coefficients of which the corresponding exponent is not divisible by d. This can be efficiently implemented based on the algorithm of Chen and Han [8]. They use the special automorphism $\tau_{n/2^i+1}$ that maps

$$\tau_{n/2^i+1} \colon x^{2^i} \mapsto -x^{2^i}.$$

That is, it negates the coefficients of $x^{2^i \cdot k}$ for odd k, and leaves the coefficients untouched for even k. This insight is used in a recursive procedure: on input a plaintext $\boldsymbol{\alpha}_0$, we compute $\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i + \tau_{n/2^i+1}(\boldsymbol{\alpha}_i)$ for $i < \log_2(d)$ until we end up with $d \cdot \boldsymbol{\alpha} = \boldsymbol{\alpha}_{\log_2(d)}$. Note that the factor of d is a result from the repeated summation. Fortunately, it can be compensated by folding an additional factor of $d^{-1} \pmod{p^e}$ in the subsequent linear transformation. Remarkably, the cost of coefficient selection is equal to the slot-wise trace map.

Finally, we map the coefficients of $\boldsymbol{\alpha}$ back into the plaintext slots. Similarly to the forward linear transformation, this can be done as

$$\boldsymbol{\alpha} \mapsto \sum_{h \in S} \boldsymbol{\kappa}_h \cdot \tau_h(\boldsymbol{\alpha}), \tag{22}$$

where the constants $\boldsymbol{\kappa}_h$ are computed by solving a system of linear equations based on ℓ independent input-output pairs. The correctness of this procedure is more difficult to prove, and given in Lemma 2.

Correctness. We now prove the correctness of the optimized linear transformations for power-of-two cyclotomics. In particular, we need to prove that the systems of linear equations to compute the constants $\boldsymbol{\kappa}_h = \kappa_h(x)$ are consistent. This is trivial for the forward linear map as it can be easily seen that the involved matrix is the identity. However, the analysis for the inverse map is more complicated, and captured by the following lemma.

Lemma 2. *If the set S is generated using the procedure from above, then the optimized coefficient-to-slot transformation for power-of-two cyclotomics can be expressed as Equation (22).*

Proof. It suffices to prove that the system of linear equations to compute the constants $\kappa_h(x)$ is consistent for each slot. First, note that the functionality of the inverse map is precisely opposite to Equation (20). Therefore, we have

$$\alpha(x) = \sum_{j=0}^{\ell-1} \alpha_{dj} x^{dj} \mapsto \beta(x) = \sum_{h \in S} \kappa_h(x) \cdot \tau_h(\alpha(x)),$$

where $\beta(x)$ contains $(\alpha_0, \ldots, \alpha_{d(\ell-1)})$ in the slots. We compute the constants $\kappa_h(x)$ by successively choosing $(\alpha_0, \ldots, \alpha_{d(\ell-1)})$ as each possible unit vector, and then solving the obtained system of linear equations for each plaintext slot. Following this procedure, the system of equations for the first slot becomes

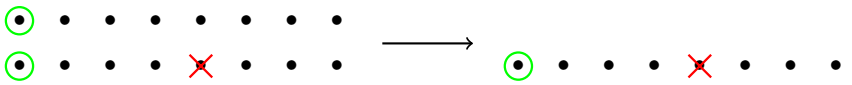
$$\begin{bmatrix} \zeta^{h_1 \cdot 0} & \cdots & \zeta^{h_\ell \cdot 0} \\ \vdots & \ddots & \vdots \\ \zeta^{h_1 \cdot d(\ell-1)} & \cdots & \zeta^{h_\ell \cdot d(\ell-1)} \end{bmatrix} \begin{bmatrix} \kappa_{h_1}(\zeta) \\ \vdots \\ \kappa_{h_\ell}(\zeta) \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix},$$

where $S = \{h_1, \ldots, h_\ell\}$. All other slots are handled analogously by replacing ζ with ζ^h for an appropriate $h \in S$.

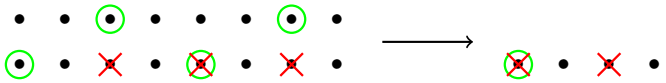
It suffices to show that the matrix from the above system (referred to as M from now on) is invertible. Note that M is a transposed Vandermonde matrix evaluated in $\zeta^{h_1 \cdot d}, \ldots, \zeta^{h_\ell \cdot d}$, and therefore known to be invertible if and only if all $\zeta^{h_i \cdot d}$ are distinct. Now suppose by contradiction that $\zeta^{h_i \cdot d} = \zeta^{h_j \cdot d}$ for $i \neq j$. Since ζ is a primitive mth root of unity, it follows that $(h_i - h_j) \cdot d = 0 \pmod{m}$, or even simpler $h_i = h_j \pmod{m/d}$. Now let $h_i = g_1^{e_1} \cdot g_2^{e_2}$ and $h_j = g_1^{f_1} \cdot g_2^{f_2}$, then the equations simplify to $g_1^{e_1 - f_1} \cdot g_2^{e_2 - f_2} = 1 \pmod{m/d}$. We need to contradict this claim for $(e_1, e_2) \neq (f_1, f_2)$ in order to finish the proof.

We can assume without loss of generality that $0 \leqslant e_1 - f_1 < 2$ and that $0 \leqslant e_2 - f_2 < \ell/2$. In particular, considering positive exponents is sufficient because negation corresponds to replacing g_1 or g_2 by its own inverse. Since the hypercube generation procedure remains valid when either generator is replaced by its inverse, negative exponents are covered by an analogous proof. With this simplification, it remains to prove that $S \setminus \{1\}$ contains no $h = 1 \pmod{m/d}$.

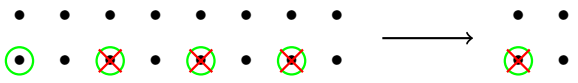
The multiplicative group $\mathbb{Z}_m^*$ has a very special structure if m is a power of two [17]. It is non-cyclic, but generated by two elements: a not further specified element g of order 2, and the element 5 of order $m/4 = n/2$. Moreover, the subgroup $\langle 5 \rangle$ contains all numbers that are congruent to 1 modulo 4. An example configuration for $m = 32$ is drawn in Figure 6. The subfigures show several possibilities for the subgroup $\langle p \rangle$ in green, and the quotient group $\mathbb{Z}_m^*/\langle p \rangle$ that is represented by S. All elements $h \in \mathbb{Z}_m^* \setminus \{1\}$ for which $h = 1 \pmod{m/d}$ are marked by a red cross. Recall that these elements are not allowed to be in S.



(a) Cyclic quotient group with p of order $d = 2$



(b) Cyclic quotient group with p of order $d = 4$



(c) Non-cyclic quotient group with p of order $d = 4$

Fig. 6: Possible configurations for the unit group $\mathbb{Z}_m^*$ and quotient group $\mathbb{Z}_m^*/\langle p \rangle$

The group law of Figure 6 should be interpreted as follows: each subfigure shows $\mathbb{Z}_m^*$ on the left, where the bottom left element with coordinates $(0, 0)$ is

identity; the element to the right of identity with coordinates $(1, 0)$ is 5; the element above identity with coordinates $(0, 1)$ is g. Multiplying two elements is done by summing their coordinates and reducing modulo the grid. Moreover, each subfigure shows how $\mathbb{Z}_m^*$ collapses into the quotient group $\mathbb{Z}_m^*/\langle p \rangle$ on the right. Each coset is indicated by exactly one representative.

In order to prove the lemma, we need to ensure that the hypercube generation procedure avoids the elements marked by red crosses. For the crosses that coincide with the subgroup $\langle p \rangle$, this is trivial because the set S always contains the representative 1 (which does not have a red cross). However, subfigures 6a and 6b have one extra cross that does not coincide with $\langle p \rangle$. In both cases, we avoid it by taking the first representative such that $g_1 - 1$ has a 2-adic valuation of 1. In other words, we choose g_1 from the upper row (which does not contain any red crosses), because those elements are congruent to 3 modulo 4.

6 Digit Extraction

Digit extraction is the essential step of bootstrapping that removes the actual noise. Since the entire procedure must be applied homomorphically, it is written out as a series of polynomial evaluations. We explain two algorithms for digit extraction: the first one is based on the procedure of Halevi and Shoup [30], and the second one is developed by Chen and Han [8]. Both procedures were later improved based on a systematic study of the involved polynomials [18]. However, those optimizations are too extensive to discuss in this paper.

The functionality is defined as follows: given a number $w \in \mathbb{Z}_{p^e}$, we remove its $v = e - r$ least significant digits. Here we always refer to the balanced digits denoted by $w_i \in \{-(p-1)/2, \ldots, (p-1)/2\}$, where we have that

$$w = \sum_{i=0}^{e-1} w_i p^i.$$

With this notation in mind, the output of digit extraction is formally defined as

$$\left\lfloor \frac{w}{p^v} \right\rceil = \sum_{i=v}^{e-1} w_i p^{i-v},$$

when given w at the input.[10]

An important cost measure of digit extraction is the multiplicative depth, i.e., the largest number of consecutive multiplications in each possible path of the evaluated circuit. For example, a polynomial of degree n can be evaluated with depth $\lceil \log_2(n) \rceil$ ciphertext-ciphertext multiplications and 1 constant-ciphertext multiplication. Based on this logarithmic relation, we can convert between the terms depth and degree.

[10] If $p = 2$, we consider the digits in $\{0, 1\}$. Hence the output of digit extraction changes to $\lfloor w/p^v \rfloor$. However, bootstrapping requires a rounding operation instead of flooring. Fortunately, this can be fixed by applying the simple equality $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$.

6.1 The Halevi/Shoup Procedure

The digit extraction procedure of Halevi and Shoup needs the following lemma, which introduces a polynomial $F_e(y)$ called the *lifting polynomial*.

Lemma 3 ([30, Corollary 5.5]). *For every prime p and exponent $e \geqslant 1$, there exists a polynomial $F_e(y) \in \mathbb{Z}[y]$ of degree p such that for all integers $1 \leqslant e' \leqslant e$, $-p/2 < w_0 \leqslant p/2$ and w_1, it holds that $F_e(w_0 + p^{e'} w_1) = w_0 \pmod{p^{e'+1}}$.*

Algorithm 1 gives the digit extraction procedure from Halevi and Shoup. Each iteration applies the lifting polynomial in order to remove one additional upper digit, and thus to extract the least significant digit. Then we use the result to compute $w_{i+j+1,0}$ via an iterative procedure consisting of (i) subtraction of the extracted digit with an adequate number of removed upper digits and (ii) exact division by p. Observe that $w_{i,j} = w_i \pmod{p^{j+1}}$ for all $i < v$ at the end of the algorithm. The cost is dominated by $ev - v(v+1)/2$ evaluations of the lifting polynomial, and the degree is p^{e-1}.

Algorithm 1 Halevi/Shoup digit extraction

```
1: procedure DigitExtract(w, p, e, v)
2:     for i ← 0 to e − 1 do
3:         w_{i,0} ← w
4:     for i ← 0 to v − 1 do
5:         for j ← 0 to e − i − 2 do
6:             w_{i,j+1} ← F_{j+1}(w_{i,j})
7:             w_{i+j+1,0} ← (w_{i+j+1,0} − w_{i,j+1})/p
8:     return w_{e−1,0}
```

6.2 The Chen/Han Procedure

The digit extraction procedure of Chen and Han needs the following lemma, which introduces a polynomial $G_e(y)$ called the *lowest digit retain polynomial*.

Lemma 4 ([8, Section 3.2]). *For every prime p and exponent $e \geqslant 1$, there exists a polynomial $G_e(y) \in \mathbb{Z}[y]$ of degree at most $(e-1)(p-1)+1$ such that for all integers $-p/2 < w_0 \leqslant p/2$, it holds that $G_e(w_0 + pw_1) = w_0 \pmod{p^e}$.*

Algorithm 2 gives the digit extraction procedure from Chen and Han. The main innovation compared to the Halevi/Shoup procedure is that we apply the lowest digit retain polynomial directly to the input, resulting in a procedure of lower multiplicative depth. This is due to the very small degree of the lowest digit retain polynomial (only $(e-1)(p-1)+1$ compared to degree p^{e-1} for $e-1$ repetitions of the lifting polynomial). However, note that we still use the lifting polynomial to compute the intermediate numbers $w_{i+j+1,0}$, because applying the

polynomial $G_{e-i}(y)$ would rather increase the multiplicative depth in this place. Observe that $w_{i,j} = w_i \pmod{p^{j+1}}$ at the end of the algorithm. The cost is dominated by v evaluations of the lowest digit retain polynomial and $v(v-1)/2$ evaluations of the lifting polynomial, and the degree is roughly rp^v.

Algorithm 2 Chen/Han digit extraction

```
1: procedure DigitExtract(w, p, e, v)
2:     z ← w
3:     for i ← 0 to v − 1 do
4:         w_{i,0} ← w
5:     for i ← 0 to v − 1 do
6:         z ← (z − G_{e−i}(w_{i,0}))/p
7:         for j ← 0 to v − i − 2 do
8:             w_{i,j+1} ← F_{j+1}(w_{i,j})
9:             w_{i+j+1,0} ← (w_{i+j+1,0} − w_{i,j+1})/p
10:    return z
```

7 Implementation and Results

This section discusses asymptotic complexities for the linear transformations and digit extraction. We also give an overview of the implementation.

7.1 Baby-Step/Giant-Step Algorithms

As already briefly explained, the linear transformations are implemented via a baby-step/giant-step algorithm. The idea is to rewrite Equation (15) as a double summation, both of size roughly $\sqrt{\ell_i}$, and bring part of the rotation outside the inner sum. The rotations of the inner sum can then be precomputed and reused in each iteration of the outer sum. Equation (17) is already in the shape of a double summation, so the trick can be applied directly in that case. As a result, linear transformations have the following time complexity:

- A one-dimensional E-linear transformation in a good hypercube dimension requires $2\sqrt{\ell_i} + \mathcal{O}(1)$ automorphisms and ℓ_i constant-ciphertext multiplications. In a bad hypercube dimension, it requires $3\sqrt{\ell_i} + \mathcal{O}(1)$ automorphisms and $2\ell_i$ constant-ciphertext multiplications.
- A one-dimensional $\mathbb{Z}_{p^r}$-linear transformation in a good hypercube dimension requires $d + \ell_i - 2$ automorphisms and $d \cdot \ell_i$ constant-ciphertext multiplications. In a bad hypercube dimension, it requires $2d + \ell_i - 3$ automorphisms and $2d \cdot \ell_i$ constant-ciphertext multiplications.

Similarly to the linear transformations, we can also evaluate polynomials using a baby-step/giant-step technique [21, 38]. Here the goal is to reduce the

number of ciphertext-ciphertext multiplications, because they are much more costly than scalar-ciphertext multiplications (with scalars from $\mathbb{Z}_{p^r}$). Specifically, we rewrite a polynomial of degree at most $n = 2^m \cdot k$ as

$$F(y) = \sum_{i=0}^{n} a_i y^i = \sum_{i=0}^{n/2-1} a_i y^i + y^{n/2} \cdot \sum_{i=0}^{n/2} a_{i+n/2} y^i.$$

This trick is applied recursively until we arrive at a set of polynomials of degree at most k. The powers of y are then precomputed and reused to compute each of these polynomials. In this process, we can choose the parameters k and m as to minimize the number of ciphertext-ciphertext multiplications. The asymptotic time complexity is then $2\sqrt{n} + \mathcal{O}(\log_2(n))$ ciphertext-ciphertext multiplications and n constant-ciphertext multiplications. The multiplicative depth is $\lceil \log_2(n) \rceil$ ciphertext-ciphertext multiplications and 1 constant-ciphertext multiplication.

7.2 Asymptotic Complexities

We can now determine the asymptotic complexities (both time and depth) of the linear transformations and digit extraction. Everything is broken down into the most costly operations: multiplications and automorphisms.

Linear transformations. The depth of the linear transformations is 1 constant-ciphertext multiplication per stage. This needs to be doubled in order to count the forward and inverse map. The total number of operations is more difficult to count. Table 1 gives the result for fully packed slots, and Table 2 for sparsely packed slots. The tables assume that we use the baby-step/giant-step algorithm. The cost depends on whether the hypercube dimension is good or bad.[11] Note that thin bootstrapping also includes one evaluation of the slot-wise trace map during the inverse transformation, which costs $\mathcal{O}(\log_2(d))$ automorphisms.

The optimized linear transformations for power-of-two cyclotomics (explained in Section 5.2) can be somewhat cheaper than the general version. Their cost is always given by the first column of Table 2, regardless of whether the hypercube dimension is good or bad. It also includes one coefficient selection step that costs exactly $\log_2(d)$ automorphisms (equal to the slot-wise trace map).

From the applications perspective, the linear transformations of Halevi and Shoup are better than the transformations from Chen and Han. The former ones can be decomposed in multiple stages, and therefore scale well for an increasing number of slots. This comes at the cost of a slight increase in multiplicative depth. Moreover, non-power-of-two cyclotomics can have a more efficient packing in the plaintext slots even for small values of p. On the other hand, many FHE implementations are restricted to power-of-two cyclotomics. In that case, the Chen/Han version is preferred, because it has no performance drawback in bad hypercube dimensions.

[11] Recall that only the first hypercube dimension can be bad. The cost for stages $2, \ldots, t$ is therefore only relevant in good hypercube dimensions.

Table 1: Total number of operations for the general linear transformations and fully packed slots

	Stage 1		Stages $2, \dots, t$
	Good	Bad	Good
Automorphism	$d + \ell_1 - 2$	$2d + \ell_1 - 3$	$2\sqrt{\ell_i} + \mathcal{O}(1)$
Multiplication	$d \cdot \ell_1$	$2d \cdot \ell_1$	ℓ_i

Table 2: Total number of operations for the general linear transformations and sparsely packed slots

	Stage 1		Stages $2, \dots, t$
	Good	Bad	Good
Automorphism	$2\sqrt{\ell_1} + \mathcal{O}(1)$	$3\sqrt{\ell_1} + \mathcal{O}(1)$	$2\sqrt{\ell_i} + \mathcal{O}(1)$
Multiplication	ℓ_1	$2\ell_1$	ℓ_i

Digit extraction. As already discussed in Section 6, the time complexity and multiplicative depth of digit extraction depend on the degrees of the lifting polynomial and the lowest digit retain polynomial. We summarize the results in Tables 3 and 4. Here we make a difference between scalar-ciphertext multiplication and ciphertext-ciphertext multiplication. The results are asymptotic (constant terms are neglected in appropriate places) and they assume that we use the baby-step/giant-step algorithm as discussed earlier.

The main advantage of Chen/Han digit extraction is in multiplicative depth. Whereas the depth of Halevi/Shoup grows linearly in the plaintext precision r, the depth of Chen/Han grows only logarithmically. This assumes that $v = e - r$ is constant, which is true for fixed m and h. Moreover, the number of operations is also different: Chen/Han is a bit more expensive for small parameters, but becomes asymptotically better due to the square root term.

Table 3: Multiplicative depth of digit extraction

	Halevi/Shoup	Chen/Han
Scalar	$e - 1$	v
Non-scalar	$(e-1) \cdot \lceil \log_2(p) \rceil$	$v \cdot \lceil \log_2(p) \rceil + \lceil \log_2(r) \rceil$

Table 4: Total number of multiplications for digit extraction

	Halevi/Shoup	Chen/Han
Scalar	$v \cdot (r + v/2) \cdot p$	$v \cdot e \cdot p$
Non-scalar	$v \cdot (r + v/2) \cdot 2\sqrt{p}$	$v \cdot (\sqrt{r + v/2} + v/2) \cdot 2\sqrt{p}$

7.3 Implementation Overview

Our high-level Magma implementation of bootstrapping is made publicly available at https://github.com/KULeuven-COSIC/Bootstrapping_BGV_BFV. Here we discuss some aspects of the implementation and the structure of the library. Our code contains the following subdirectories:

- `CRT`: this folder contains functions to convert in and out of Double-CRT format [24]. It is only used for automorphisms, because all other operations can be handled directly via built-in functionality.
- `Crypto`: this folder contains the implementation of BGV and BFV. It also contains an essential file `Params.m` where the recryption parameters are set. The implementation does not leverage Double-CRT and RNS data representation (except for automorphisms), but uses built-in polynomial operations based on multiprecision arithmetic instead.
- `Linear maps`: this folder contains the implementation and specification of the linear transformations. We include both the general version and the optimized one for power-of-two cyclotomics.
- `Digit extraction`: this folder contains the Halevi/Shoup and Chen/Han digit extraction procedures. It also implements that baby-step/giant-step technique necessary for polynomial evaluation.
- `Bootstrapping`: this folder contains the general and thin recryption procedures. Switching from BGV to BFV or vice versa can simply be done by loading a different scheme.

8 Conclusion

Gentry's bootstrapping technique remains a costly operation in the construction of fully homomorphic encryption schemes. This article revisited bootstrapping for BGV and BFV, two second generation schemes that support modular arithmetic over their plaintext space. Both have an identical recryption procedure, and are only distinguished by small differences in the implementation of the underlying operations. Our observations are supported by a high-level implementation in Magma. It is made publicly available, and it can be used by application developers who want to represent bootstrapping from a high level.

Our comparative study considers several versions of the linear maps involved in bootstrapping. The Halevi/Shoup version is in general more efficient, because it can be decomposed as a series of one-dimensional linear transformations. On the other hand, the Chen/Han version can be slightly more performant when restricted to power-of-two cyclotomics.

In terms of digit extraction, the procedure of Chen and Han is asymptotically better for high exponents in the plaintext modulus. The advantage appears both in time complexity and multiplicative depth. On the other hand, the procedure of Halevi and Shoup requires less multiplications for a low-precision plaintext space. We propose that future work concentrates on the digit extraction step. This remains the bottleneck of bootstrapping because of the high number of key switching operations involved.

Acknowledgements. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-21-C-0034. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This work was additionally supported in part by CyberSecurity Research Flanders with reference number VR20192203. Robin Geelen is funded in part by Research Foundation – Flanders (FWO) under a PhD Fellowship fundamental research (project number 1162123N). Finally, the authors would like to thank Kyoohyung Han and Yuriy Polyakov for their review and Steven Galbraith for spotting an error in an earlier version of this paper.

References

1. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Annual Cryptology Conference. pp. 1–20. Springer (2013)
2. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915, https://eprint.iacr.org/2022/915
3. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 587–617. Springer International Publishing, Cham (2021)
4. Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. Cryptology ePrint Archive, Paper 2022/024 (2022), https://eprint.iacr.org/2022/024, https://eprint.iacr.org/2022/024
5. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Simulating homomorphic evaluation of deep learning predictions. In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 212–230. Springer (2019)
6. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ITCS '12, ACM (2012)
8. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–337. Springer (2018)
9. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1223–1237 (2018)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421 (2016), https://eprint.iacr.org/2016/421, https://eprint.iacr.org/2016/421

11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Paper 2016/870 (2016), https://eprint.iacr.org/2016/870, https://eprint.iacr.org/2016/870
12. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European transactions on Telecommunications **8**(5), 481–490 (1997)
13. Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion oram: A constant bandwidth blowup oblivious ram. In: Theory of Cryptography Conference. pp. 145–174. Springer (2016)
14. Dong, C., Chen, L.: A fast single server private information retrieval protocol with low communication cost. In: Computer Security - ESORICS 2014, Lecture Notes in Computer Science, vol. 8712, pp. 380–399. Springer International Publishing, Cham (2014)
15. Ducas, L., Micciancio, D.: Fhew: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Paper 2014/816 (2014), https://eprint.iacr.org/2014/816, https://eprint.iacr.org/2014/816
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
17. Gauss, C.F.: Disquisitiones arithmeticae. Springer, Berlin (1986)
18. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. Cryptology ePrint Archive, Paper 2022/1364 (2022), https://eprint.iacr.org/2022/1364, https://eprint.iacr.org/2022/1364
19. Geelen, R., Van Beirendonck, M., Pereira, H.V., Huffman, B., McAuley, T., Selfridge, B., Wagner, D., Dimou, G., Verbauwhede, I., Vercauteren, F., et al.: Basalisc: Flexible asynchronous hardware accelerator for fully homomorphic encryption. arXiv preprint arXiv:2205.14017 (2022)
20. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
21. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 129–148. Springer (2011)
22. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: International Workshop on Public Key Cryptography. pp. 1–16. Springer (2012)
23. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Advances in Cryptology – EUROCRYPT 2012, Lecture Notes in Computer Science, vol. 7237, pp. 465–482. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
24. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
25. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Paper 2013/340 (2013), https://eprint.iacr.org/2013/340, https://eprint.iacr.org/2013/340
26. Halevi, S., Shoup, V.: Algorithms in helib. Cryptology ePrint Archive, Report 2014/106 (2014), https://eprint.iacr.org/2014/106

27. Halevi, S., Shoup, V.: Bootstrapping for helib. In: Annual International conference on the theory and applications of cryptographic techniques. pp. 641–670. Springer (2015)
28. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in helib. In: Annual International Cryptology Conference. pp. 93–120. Springer (2018)
29. Halevi, S., Shoup, V.: Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481 (2020), https://eprint.iacr.org/2020/1481
30. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(1), 1–44 (2021)
31. Jutla, C.S., Manohar, N.: Sine series approximation of the mod function for bootstrapping of approximate he. Cryptology ePrint Archive, Paper 2021/572 (2021), https://eprint.iacr.org/2021/572, https://eprint.iacr.org/2021/572
32. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for rlwe-based homomorphic encryption. Cryptology ePrint Archive, Paper 2021/691 (2021), https://eprint.iacr.org/2021/691, https://eprint.iacr.org/2021/691
33. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 608–639. Springer (2021)
34. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 618–647. Springer International Publishing, Cham (2021)
35. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 551–580. Springer International Publishing, Cham (2022)
36. Li, R., Jia, C.: Homomorphic modular reduction and improved bootstrapping for bgv scheme. In: Information Security and Cryptology, Lecture Notes in Computer Science, vol. 13007, pp. 466–484. Springer International Publishing, Cham (2021)
37. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 35–54. Springer (2013)
38. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973)
39. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
40. Roman, S.: Field theory, vol. 158. Springer Science & Business Media (2005)
41. Rondeau, T.: Data protection in virtual environments (DPRIVE) (2020)
42. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**(1), 57–81 (2014)
43. Zucca, V.: Towards efficient arithmetic for ring-lwe based homomorphic encryption (2018)