On Polynomial Functions Modulo p^e and Faster Bootstrapping for Homomorphic Encryption

Robin Geelen[1], Ilia Iliashenko[2], Jiayi Kang[1], and Frederik Vercauteren[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] CipherMode Labs, Los Angeles, USA
ilia@ciphermode.com

Abstract. In this paper, we perform a systematic study of functions $f : \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e}$ and categorize those functions that can be represented by a polynomial with integer coefficients. More specifically, we cover the following properties: necessary and sufficient conditions for the existence of an integer polynomial representation; computation of such a representation; and the complete set of equivalent polynomials that represent a given function.
As an application, we use the newly developed theory to speed up bootstrapping for the BGV and BFV homomorphic encryption schemes. The crucial ingredient underlying our improvements is the existence of null polynomials, i.e. non-zero polynomials that evaluate to zero in every point. We exploit the rich algebraic structure of these null polynomials to find better representations of the digit extraction function, which is the main bottleneck in bootstrapping. As such, we obtain sparse polynomials that have 50% fewer coefficients than the original ones. In addition, we propose a new method to decompose digit extraction as a series of polynomial evaluations. This lowers the time complexity from $\mathcal{O}(\sqrt{pe})$ to $\mathcal{O}(\sqrt{p}\sqrt[4]{e})$ for digit extraction modulo p^e, at the cost of a slight increase in multiplicative depth. Overall, our implementation in `HElib` shows a significant speedup of a factor up to 2.6 over the state-of-the-art.

Keywords: Homomorphic encryption · Bootstrapping · Polyfunctions.

1 Introduction

Homomorphic encryption (HE) allows computations on encrypted data without knowledge of the secret key. In the past 15 years, there have been tremendous improvements in HE protocols, both in speed and applicability. In spite of these efforts, homomorphic encryption remains extremely slow compared to unencrypted computations and further speedups are required.

Homomorphic computations are typically realized as arithmetic circuits, i.e. sequences of additions and multiplications that implement a desired functionality. In the lattice-based schemes BGV [6] and BFV [5, 10], these operations are performed over (extensions of) $\mathbb{Z}_{p^e}$, where p is a prime number and e is

a positive integer.[3] Functions on $\mathbb{Z}_{p^e}$ have rather interesting properties. First, only a limited class of functions can be described by polynomials with integer coefficients, the so-called *polyfunctions* (short for polynomial functions). Second, the polynomials that represent a given polyfunction are always non-unique and we can therefore try to find the polynomial representation that is most efficient to evaluate homomorphically.

An example application that can benefit from the study of polyfunctions is *bootstrapping* – the ciphertext refreshing procedure that enables unbounded fully homomorphic encryption. This procedure is necessary because lattice-based schemes include a noise term that grows when we evaluate an arithmetic circuit. Bootstrapping reduces the noise back to a lower level, which enables further evaluation of homomorphic additions and multiplications. Since its introduction by Gentry in 2009 [11], the latency and throughput of bootstrapping were improved several orders of magnitude in many subsequent works [8, 13, 15], but it remains the main bottleneck to achieve fully homomorphic encryption.

1.1 Related Work

Polyfunctions. Research into polyfunctions has a long history. Already in 1921, Kemper [17] studied elementary structures of polyfunctions over $\mathbb{Z}_m$ for a composite integer m. This early research represents polynomials in the monomial basis $\{X^i\}_{i=0,1,\ldots}$. However, since the mid-1960s, much of the literature [7,9,16,23] started to use the falling factorial basis $\{X \cdot (X-1) \cdot \ldots \cdot (X-i)\}_{i=0,1,\ldots}$. The reason for this shift is that the falling factorial polynomials almost directly give rise to non-trivial *null polynomials* (i.e. polynomials that by definition evaluate to zero in every point when interpreted modulo some prime power p^e).

Null polynomials result in equivalent representations of the same polyfunction $f\colon \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e}$. Specifically, two polynomials $F(X), H(X) \in \mathbb{Z}[X]$ represent the same function f if and only if their difference $F(X) - H(X)$ is a null polynomial. Equivalently, the set of all possible representations of f is obtained as $F(X) + \mathcal{O}_{p^e}$, where $\mathcal{O}_{p^e}$ is the set of all null polynomials modulo p^e. In other words, there exists a one-to-one correspondence between polyfunctions and collections of equivalent polynomials:

$$\text{polyfunction } f\colon \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e} \iff F(X) + \mathcal{O}_{p^e}.$$

Bootstrapping. The first bootstrapping procedure for BGV was proposed by Gentry et al. [13] for encryption of single bits, and improved by subsequent research [1]. The most relevant works for this paper are from Halevi and Shoup [15], and Chen and Han [8]. Halevi and Shoup proposed a bootstrapping method that works for the more general plaintext space $\mathbb{Z}_{p^e}$. Their technique relies on a "digit removal" procedure, which involves repeated homomorphic evaluation of

[3] Some protocols for secure multi-party computation [3] also work over $\mathbb{Z}_{p^e}$, which makes our study of polyfunctions even more widely applicable. However, improvements in multi-party computation are not the direct focus of this paper.

the *lifting polynomial* and has degree p^{e-1} in total. Chen and Han introduced an additional *digit extraction polynomial* (sometimes called the *lowest digit retain polynomial*) that has a much lower degree equal to $(p-1)\cdot(e-1)+1$. Lower degrees are typically favored in homomorphic encryption.

In practice, polynomial evaluations account for most of the computational cost of bootstrapping: in the implementation of `HElib`, they are altogether $3\times$ to $50\times$ more expensive than all other operations combined [15]. This situation is exactly the same for BGV and BFV, because both schemes have an identical bootstrapping procedure.

1.2 Our Contributions

The aim of this paper is to further develop the theory of polyfunctions with a focus on cryptographic applications. New insights in these polyfunctions allow us to significantly accelerate HE bootstrapping.

Polyfunctions. In the first part of the paper (Section 3), we study polyfunctions modulo p^e. This includes the following:

- In Section 3.1, we study the complete set of null polynomials modulo p^e (denoted by $\mathcal{O}_{p^e}$) as to obtain the set of all equivalent polyfunction representations. A novel element of our approach is also restricting $\mathcal{O}_{p^e}$ to contain only polynomials of bounded degree. When doing so, the resulting set forms a lattice structure, and we can find small-coefficient representations by solving the closest vector problem in this lattice. This is interesting in homomorphic encryption, because small coefficients lead to less noise growth.
- In Section 3.3, we extend Newton interpolation from the real numbers to $\mathbb{Z}_{p^e}$. Our method always returns a polynomial representation of the lowest degree when given a polyfunction as input. When given a function that is not a polyfunction, our method can detect this and returns an error.
- In Section 3.5, we discuss several properties of polyfunctions that are especially relevant for HE bootstrapping. In particular, we consider the class of even and odd polyfunctions that satisfy respectively $f(-a) = f(a) \pmod{p^e}$ and $f(-a) = -f(a) \pmod{p^e}$ for $a \in \mathbb{Z}$. We show that each such function can be represented by a sparse polynomial with only even- or odd-exponent terms. Evaluating such a sparse representation is asymptotically cheaper by a factor of $\sqrt{2}$.

Bootstrapping. In the second part of the paper (Sections 4 and 5), we apply the newly developed theory to speed up BGV and BFV bootstrapping. The most expensive component of bootstrapping, both in degree and execution time, is evaluation of the digit extraction polynomial. In order to accelerate it, we apply the following improvements:

- We propose multiple methods to obtain better representations of the digit extraction function. First, we show that this function is either even or odd,

and can therefore be represented as a polynomial with only 50% of the coefficients. Second, we propose a new technique to decompose digit extraction in multiple stages. Let g_e be the digit extraction function modulo p^e, then we write it as $g_e = g_{e,e'} \circ g_{e'}$. In our algorithm, both $g_{e'}$ and $g_{e,e'}$ are evaluated using polynomials of much smaller degree than the direct approach. As a consequence, we lower the time complexity for digit extraction from $\mathcal{O}(\sqrt{pe})$ to $\mathcal{O}(\sqrt{p}\sqrt[4]{e})$, at the cost of $\lceil \log_2 p \rceil$ increase in multiplicative depth.
- In order to fully benefit from the optimized digit extraction polynomials, we revise the digit removal procedure of Chen and Han [8]. Our improved algorithm utilizes the digit extraction polynomial exclusively, without relying on the lifting polynomial. We implemented our new bootstrapping algorithm in `HElib`, and observe that it is up to 2.6 times faster than the state-of-the-art. Our code is made publicly available.[4]

2 Preliminaries

2.1 Notations

For prime p and integer exponent $e \geqslant 1$, the set of functions from $\mathbb{Z}_{p^e}$ to itself is denoted by $\mathcal{F}_{p^e}$. Moreover, we write the evaluation of a polynomial $F(X)$ at $X = a$ as $F(a)$ or sometimes $F(X)|_{X=a}$.

Let $\nu_p(\cdot)$ denote the *p-adic valuation* function defined as

$$\nu_p(m) = \begin{cases} \max\{k \in \mathbb{N} : p^k \mid m\} & \text{if } m \neq 0 \\ \infty & \text{if } m = 0. \end{cases}$$

It generalizes to the rational numbers as $\nu_p(m/n) = \nu_p(m) - \nu_p(n)$, and we call a rational number *p-integral* if its p-adic valuation is non-negative. Let $\mu(\cdot)$ denote the *Smarandache* function defined as

$$\mu(k) = \min\{i \in \mathbb{N} : k \mid i!\}\,.$$

Observe that $\nu_p(\cdot)$ and $\mu(\cdot)$ are complementary in some sense. Specifically, it follows directly from the above definitions that $\mu(p^e)$ is the smallest integer for which $\nu_p(\mu(p^e)!) \geqslant e$. A few example instances of $\nu_p(n!)$ and $\mu(p^e)$ for $p = 2$ are listed in Tables 1 and 2.

Table 1: Examples of $\nu_2(n!)$

n	1	2	3	4	5	6	7	8	9	10
$\nu_2(n!)$	0	1	1	3	3	4	4	7	7	8

Table 2: Examples of $\mu(2^e)$

e	1	2	3	4	5	6	7	8	9	10
$\mu(2^e)$	2	4	4	6	8	8	8	10	12	12

[4] See https://github.com/KULeuven-COSIC/Bootstrapping_Polyfunctions.

2.2 Newton Interpolation over $\mathbb{R}$

The Falling Factorial Basis. The Newton interpolation method relies on the so-called *falling factorial polynomials*. Those polynomials are indexed by an integer $i \geqslant 0$ and defined as

$$(X)_i = \prod_{k=0}^{i-1} (X-k) \in \mathbb{Z}[X],$$

where by definition we set $(X)_0 = 1$. When reduced modulo p^e, these polynomials exhibit very specific properties that will be studied later in this paper.

Let $\mathbb{P}_n \subseteq \mathbb{Z}[X]$ be the set of polynomials of degree at most n. Obviously, the set $\{X^i \mid 0 \leqslant i \leqslant n\}$ forms a basis for $\mathbb{P}_n$ when seen as a module over $\mathbb{Z}$. We refer to it as the monomial basis. Similarly, also the set $\{(X)_i \mid 0 \leqslant i \leqslant n\}$ forms a basis for $\mathbb{P}_n$, known as the falling factorial basis.

Newton Interpolation. Consider a collection of $n+1$ data points $(i, y_i) \in \mathbb{R}^2$ for $i = 0, \ldots, n$.[5] Using Newton interpolation, we can find a polynomial $F(X)$ of degree at most n that interpolates these data points. Concretely, write the polynomial $F(X) \in \mathbb{R}[X]$ in the format

$$F(X) = c_0 + c_1(X)_1 + c_2(X)_2 + \ldots + c_n(X)_n. \tag{1}$$

Then we can uniquely determine the falling factorial coefficients c_i such that

$$F(i) = y_i, \quad \forall 0 \leqslant i \leqslant n.$$

The coefficients can be computed from forward differences, as introduced in the following definition.

Definition 1. *The* i-th forward difference *of a function* $f\colon \mathbb{R} \to \mathbb{R}$*, evaluated at* $j \in \mathbb{Z}$*, is recursively defined as*

$$\Delta^i f(j) = \begin{cases} f(j) & \text{if } i = 0 \\ \Delta^{i-1} f(j+1) - \Delta^{i-1} f(j) & \text{if } i > 0. \end{cases}$$

We will now apply these forward differences to a polynomial $F(X)$. Note that we slightly abuse notation and consider a polynomial as a function in X. As shown in Figure 1, the value of $\Delta^i F(X)|_{X=j}$ for $i, j = 0, 1, \ldots, n$ can be derived from Definition 1. Each element in this triangle is defined as $\alpha_{i,j} = \Delta^i F(X)|_{X=j}$, and computed as the difference between the element above and the element above left. We only show rows for $i = 0, \ldots, n$, because all following rows are zero for a polynomial of degree n. This is easily seen by computing

$$\begin{aligned} \Delta\,(X)_i &= (X+1)X \cdot \ldots \cdot (X-i+2) - X(X-1) \cdot \ldots \cdot (X-i+1) \\ &= i(X)_{i-1}, \end{aligned} \tag{2}$$

[5] In a more general version, we could consider the data points (x_i, y_i). For our purpose, however, it is sufficient to choose $x_i = i$.

and using the result in Equation (1). Note that Equation (2) is the analogue of taking the derivative of the monomial X^i.

The coefficients of the interpolating polynomial $F(X)$ can now be computed as $c_i = \alpha_{i,0} = \Delta^i F(X)|_{X=0}/i!$. This result is achieved by taking the i-th forward difference of both sides of Equation (1), and again filling in Equation (2). This leads to the interpolating polynomial

$$F(X) = \alpha_{0,0} + \alpha_{1,0}(X)_1 + \frac{\alpha_{2,0}}{2!}(X)_2 + \cdots + \frac{\alpha_{n,0}}{n!}(X)_n. \tag{3}$$

Note the analogy with the Taylor series of a function.

Finally, the following relations are useful:

$$\alpha_{0,j} = \sum_{v=0}^{j} \binom{j}{v} \alpha_{v,0}, \tag{4a}$$

$$\alpha_{i,0} = \sum_{v=0}^{i} (-1)^{i+v} \binom{i}{v} \alpha_{0,v}. \tag{4b}$$

These equations establish a relationship between the elements in the first row and the diagonal of Figure 1.

$$\begin{array}{ccccc} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \cdots & \alpha_{0,n} \\ & \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ & & \alpha_{2,0} & \cdots & \alpha_{2,n-2} \\ & & & \ddots & \vdots \\ & & & & \alpha_{n,0} \end{array}$$

Fig. 1: Evaluation of forward differences with $\alpha_{i,j} = \Delta^i F(X)|_{X=j}$.

The above theory generalizes directly to polynomial rings over any field. However, the subject of this paper is polynomials over $\mathbb{Z}_{p^e}$, which is not a field in general.

2.3 Polyfunctions Modulo p^e

Definition 2. *Let $f \in \mathcal{F}_{p^e}$ be a function from $\mathbb{Z}_{p^e}$ to itself. If there exists a polynomial $F(X) \in \mathbb{Z}[X]$ that satisfies $F(a) = f(a) \pmod{p^e}$ for all $a \in \mathbb{Z}$, then f is a* polyfunction modulo p^e *and $F(X)$ is a* representation *of f.*[6]

[6] We define the evaluation of a function $f \in \mathcal{F}_{p^e}$ at an integer a in the natural way, by implicitly converting a to its residue class modulo p^e.

As a corollary of the theory in Section 2.2, all functions from the field $\mathbb{F}_p$ to itself are polyfunctions. A unique representation of degree less than p is obtained by starting from all data points and applying Newton interpolation. However, the situation is different for functions modulo p^e: first, not all functions are described by integer polynomials modulo p^e, regardless of the degree; second, polyfunctions always have a non-unique representation of the lowest degree due to the existence of *null polynomials* [16, 19, 21, 23].

Null Polynomials Modulo p^e. We define a null polynomial as follows.

Definition 3. *An element $O(X) \in \mathbb{Z}[X]$ is called a* null polynomial modulo p^e *if the function $f \in \mathcal{F}_{p^e}$ that it represents maps every element to zero. In other words, we have that $O(a) = 0 \pmod{p^e}$ for all $a \in \mathbb{Z}$.*

Observe that the evaluation of the falling factorial polynomial $(X)_i$ at *any* integer is divisible by $i!$. Hence it is a null polynomial modulo p^e if $\nu_p(i!) \geqslant e$. Also the other direction holds: if $(X)_i$ is a null polynomial modulo p^e, then evaluating it at $X = i$ gives $(X)_i|_{X=i} = i!$, and therefore $\nu_p(i!) \geqslant e$. Following the notation defined earlier, we find that the smallest possible value of i for which $(X)_i$ is a null polynomial modulo p^e, is equal to $i = \mu(p^e)$.

2.4 Lattices

Definition 4. *The set $\mathcal{L} \subseteq \mathbb{R}^n$ is a* lattice *if there exist $\mathbb{R}$-linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{R}^n$ such that*

$$\mathcal{L} = \left\{ \sum_{i=1}^{k} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}.$$

The set of vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ constitute a basis*, and k is called the* rank*. A lattice is called* q-ary *for an integer q if $q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n$.*

For a lattice vector $\mathbf{v} \in \mathcal{L}$, the length $\|\mathbf{v}\|$ denotes its Euclidean norm (2-norm). We will rely on the closest vector problem (CVP):

Definition 5 (Closest vector problem (exact form)). *Consider a lattice $\mathcal{L} \subseteq \mathbb{R}^n$ and a vector $\mathbf{t} \in \mathbb{R}^n$, CVP asks to recover a lattice vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{t} - \mathbf{v}\| = \min_{\mathbf{y} \in \mathcal{L}} \|\mathbf{t} - \mathbf{y}\|$.*

Lattices have been studied extensively in cryptography due to the conjectured intractability of certain lattice problems, such as the shortest vector problem (SVP) and the closest vector problem (CVP). The hardness of these problems is used as the security foundation of many cryptosystems, including the BGV and BFV schemes. However, we will use lattices for a different reason, namely the study of polynomial representations with small coefficients.

2.5 Homomorphic Encryption

We are interested in homomorphic encryption schemes that support arithmetic circuits over $\mathbb{Z}_{p^e}$. In the literature, those schemes are known as BGV [6] and BFV [5, 10]. Both schemes have the same interface, and only differ from each other in terms of the underlying implementation.

Homomorphic Operations. Next to the usual key generation, encryption and decryption, homomorphic encryption schemes have two extra procedures to evaluate additions and multiplications over the ciphertexts that they encrypt. Both procedures can either take two ciphertexts, or one ciphertext and one plaintext. Moreover, there is one special division operation, which takes a ciphertext that encrypts a message m known to be divisible by p. It outputs a new ciphertext that encrypts m/p, but under plaintext modulus p^{e-1} instead of p^e. This operation fails if the input message is not divisible by p.

Plaintext Batching. BGV and BFV can batch multiple elements of $\mathbb{Z}_{p^e}$ per plaintext [22]. Specifically, the plaintext ring is isomorphic to $\mathbb{Z}_{p^e}^{\ell}$, where addition and multiplication are defined component-wise. Each copy of $\mathbb{Z}_{p^e}$ is called a *plaintext slot*, and can be operated on homomorphically and in parallel. This is sometimes referred to as SIMD operations due to the resemblance in parallel computing architectures.

The above explanation is actually a special case of a more general technique. Given a polynomial $F(X) \in \mathbb{Z}[X]$ that is irreducible modulo p, we can define the Galois ring $E = \mathbb{Z}_{p^e}[X]/(F(X)) \supseteq \mathbb{Z}_{p^e}$. The plaintext rings of BGV and BFV are then isomorphic to E^{ℓ}, again with component-wise addition and multiplication. We refer to this more general version as fully packed slots. If the slots are restricted to encode elements from the subring $\mathbb{Z}_{p^e}$ (like explained above), then they are called sparsely packed.

Bootstrapping. Every HE ciphertext contains a special component called the *noise*. When evaluating homomorphic additions and multiplications, the noise gets larger depending on the complexity of the involved operations. The decryption function removes the noise, but only works correctly if the noise is small enough (depending on the chosen scheme parameters).

To enable circuits that consist of an unlimited number of additions and multiplications, we need a method to reduce the ciphertext noise without decrypting directly. This is achieved via bootstrapping. The idea is to decrypt a ciphertext *homomorphically* by evaluating the scheme's own decryption circuit. This reduces noise and allows further evaluation of additions and multiplications. Bootstrapping comes in two variants: the slots of the encrypted message can either be fully packed or sparsely packed. We refer to the first situation as general bootstrapping, and the second one as thin bootstrapping. Finally, we emphasize that BGV and BFV have an identical bootstrapping procedure. All optimizations for one scheme therefore carry over to the other one immediately.

3 Systematic Study of Polyfunctions

3.1 Null Polynomials

The set of null polynomials modulo p^e can be described in the falling factorial basis. This was already noticed by Singmaster [21] who proved the general structure of this set. We formulate an adapted version in the following theorem, where we additionally take into account null polynomials of bounded degree. Our theorem is proven based on the same outline as Singmaster's proof.

Theorem 1. *A polynomial $O(X) \in \mathbb{Z}[X]$ is a null polynomial modulo p^e of degree at most n if and only if there exist $a_0, \dots a_n \in \mathbb{Z}$ such that*

$$O(X) = \sum_{i=0}^{n} a_i \cdot O_i(X), \quad \textit{with} \quad O_i(X) = p^{\max(e-\nu_p(i!),0)} \cdot (X)_i \,. \tag{5}$$

In this equation, the exponent of p equals 0 if $i \geqslant \mu(p^e)$.

Proof. ($\Leftarrow$) As already pointed out in Section 2.3, the evaluation of $(X)_i$ at any integer is divisible by $p^{\nu_p(i!)}$. Therefore, each term in Equation (5) evaluated at any integer is divisible by $p^{\max(e,\nu_p(i!))} \geqslant p^e$. Since each term is a null polynomial modulo p^e, so is their linear combination.

($\Rightarrow$) We prove the following assertion for $0 \leqslant m \leqslant n+1$ by applying induction on m:

$$O(X) = \sum_{i=m}^{n} b_i \cdot (X)_i + \sum_{i=0}^{m-1} a_i \cdot O_i(X), \tag{6}$$

for some $a_i, b_i \in \mathbb{Z}$.

The base case $m = 0$ is trivial since the second sum is empty, and the first sum amounts to writing a polynomial in the falling factorial basis. It is therefore possible to find appropriate constants b_i that satisfy Equation (6).

Now suppose that Equation (6) was established for some $m < n+1$, that is

$$O(X) = b_m \cdot (X)_m + \sum_{i=m+1}^{n} b_i \cdot (X)_i + \sum_{i=0}^{m-1} a_i \cdot O_i(X).$$

Evaluating both sides at $X = m$ gives

$$0 = O(m) = b_m \cdot m! \pmod{p^e}.$$

Taking the p-adic valuation of the right-hand side gives

$$\nu_p(b_m \cdot m!) = \nu_p(b_m) + \nu_p(m!) \geqslant e \implies \nu_p(b_m) \geqslant e - \nu_p(m!).$$

The constants b_i are integers, so it follows that $\nu_p(b_m) \geqslant \max(e - \nu_p(m!), 0)$. We can therefore write $b_m = a_m \cdot p^{\max(e-\nu_p(m!),0)}$ for some $a_m \in \mathbb{Z}$, which results in

$$O(X) = \sum_{i=m+1}^{n} b_i \cdot (X)_i + \sum_{i=0}^{m} a_i \cdot O_i(X).$$

This expression replaces m by $m+1$ in Equation (6) and thereby completes the induction. The final result follows by setting $m = n+1$ in Equation (6). □

Corollary 1. *Each null polynomial modulo p^e of degree $n < \mu(p^e)$ is divisible by $p^{e-\nu_p(n!)}$, where divisibility is defined in the polynomial ring $\mathbb{Z}[X]$. Therefore, all monic null polynomials have degree at least $\mu(p^e)$.*

Corollary 2. *The set of all null polynomials modulo p^e is obtained directly from Theorem 1 by allowing an arbitrarily large (but finite) degree n.*

The Null Lattice. Adopting the notation from Equation (5), the set of null polynomials of degree at most n is given by

$$\mathcal{O}_{p^e}^{(n)} = \left\{ \sum_{i=0}^{n} a_i \cdot O_i(X) \mid a_i \in \mathbb{Z} \right\} \subseteq \mathbb{P}_n.$$

When considering polynomials as coefficient vectors, it can easily be seen that the above set forms a p^e-ary lattice with basis vectors $O_i(X)$. For convenience of notation, we will not make a difference between polynomials and lattice vectors: the set $\mathcal{O}_{p^e}^{(n)}$ inherits all properties from Section 2.4, including the norm.

3.2 Cosets of Equivalent Polynomials

A representation $F(X)$ of a polyfunction f is never unique. That is, given a null polynomial $O(X)$, we can construct an equivalent polynomial $H(X) = F(X) + O(X)$ that represents the same polyfunction. The set of all representations of a polyfunction forms the coset $F(X) + \mathcal{O}_{p^e}$. Moreover, the set of all representations of degree at most n forms the coset $F(X) + \mathcal{O}_{p^e}^{(n)}$ (assuming that $\deg(F) \leqslant n$).

As explained in Section 2.3, $(X)_{\mu(p^e)}$ is a monic null polynomial modulo p^e, which implies that we can always divide by it (using Euclidean division) to obtain a representation of degree less than $\mu(p^e)$. This proves the following lemma.

Lemma 1 (Small degree representation [16]). *Each polyfunction $f \in \mathcal{F}_{p^e}$ has a representation of degree strictly less than $\mu(p^e)$.*

Although Euclidean division always returns a representation of degree less than $\mu(p^e)$, it is not necessarily minimized. In order to guarantee the lowest possible degree, one has to consecutively divide by $O_i(X)$ for $i = \mu(p^e), \ldots, 0$. This leads to the canonical representation of Keller and Olson [16].

Theorem 2 (Canonical representation [16]). *Let $f \in \mathcal{F}_{p^e}$ be a polyfunction, then there exists a unique canonical representation*

$$F(X) = \sum_{i=0}^{\mu(p^e)-1} c_i (X)_i$$

with $0 \leqslant c_i < p^{e-\nu_p(i!)}$.

From Theorem 2, we can compute the number of canonical representations, which is equal to the number of polyfunctions modulo p^e. This is done by adding the total number of possibilities for the coefficients c_i, which gives

$$\mathrm{vol}\left(\mathcal{O}_{p^e}^{(\mu(p^e)-1)}\right) = \exp_p\left(\sum_{k=0}^{\mu(p^e)-1} e - \nu_p(k!)\right) = \exp_p\left(\sum_{k=1}^{e} \mu(p^k)\right), \quad (7)$$

where $\mathrm{vol}(\cdot)$ denotes the volume of a lattice and $\exp_p(\cdot)$ the exponential function with base p. The first equality highlights the one-to-one correspondence between polyfunctions and equivalent representations of degree less than $\mu(p^e)$, obtained as cosets modulo the null lattice. The second equality was proven by Specker et al. [23] and not repeated here for brevity.

Note that, although a canonical representative can be chosen in a unique manner, it is not necessarily the most convenient polynomial to evaluate homomorphically. Later we study the digit extraction polynomial in FHE bootstrapping, where we take a different representative than the canonical choice.

Finally, we compare the number of functions in $\mathcal{F}_{p^e}$ to the number of polyfunctions from Equation (7). Since a function is uniquely determined by its input-output pairs, the total number of functions equals $(p^e)^{p^e} = p^{e \cdot p^e}$. This expression is typically much larger than Equation (7) for $e \geqslant 2$, so only very few functions are representable by polynomials.

Example 1. There are $2^{8 \cdot 2^8} \approx 10^{617}$ functions in $\mathcal{F}_{2^8}$, while only $2^{50} \approx 10^{15}$ of them are polyfunctions as computed from Equation (7).

3.3 Existence of Polynomial Representation

In this section, we examine whether a given function $f \in \mathcal{F}_{p^e}$ is a polyfunction or not. We extend the Newton interpolation method to functions modulo p^e, and return a representation of the lowest degree if f is a polyfunction.

Consider a function $f \in \mathcal{F}_{p^e}$ that is defined by p^e data points $(i, f(i)) \in \mathbb{Z}_{p^e}^2$ for $i = 0, \ldots, p^e - 1$. We will now use *reduced* forward differences, which are similar to the regular forward difference defined earlier, but include an extra reduction modulo p^e in the set $\{0, \ldots, p^e - 1\}$.

Definition 6. *The* reduced i-th forward difference *of a function $f \in \mathcal{F}_{p^e}$, evaluated at $j \in \mathbb{Z}$, is defined as*

$$\overline{\Delta^i} f(j) = \Delta^i f(j) \pmod{p^e}.$$

The values $\overline{\Delta^i} f(j)$ for $i = 0, 1, \ldots, \mu(p^e)$ and $j = 0, \ldots, p^e - 1$ can be derived from Definition 6. This is shown in Figure 2, where $\alpha_{i,j} = \overline{\Delta^i} F(X)|_{X=j}$.

The relations in Section 2.2 such as Equations (2), (4a), and (4b) still hold modulo p^e. Moreover, we will show later that the interpolating polynomial from Equation (3) is also valid for polyfunctions over $\mathbb{Z}_{p^e}$.

$$\begin{array}{ccccccc}
\alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \cdots & \alpha_{0,\mu(p^e)} & \cdots & \alpha_{0,p^e-1} \\
 & \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,\mu(p^e)-1} & \cdots & \alpha_{1,p^e-2} \\
 & & \alpha_{2,0} & \cdots & \alpha_{2,\mu(p^e)-2} & \cdots & \alpha_{2,p^e-3} \\
 & & & \ddots & \vdots & & \vdots \\
 & & & & \alpha_{\mu(p^e),0} & \cdots & \alpha_{\mu(p^e),p^e-\mu(p^e)-1}
\end{array}$$

Fig. 2: Evaluation of reduced forward differences with $\alpha_{i,j} = \overline{\Delta^i} F(X)|_{X=j}$.

Polynomial Representation. In order to examine if a function $f \in \mathcal{F}_{p^e}$ is a polyfunction, we introduce a new lemma.

Lemma 2. *Let $F(X) \in \mathbb{Q}[X]$ be a polynomial of degree less than $\mu(p^e)$ with evaluation function f. Then f interpreted modulo p^e is a polyfunction if and only if the coefficients of $F(X)$ are p-integral.*

Proof. ($\Leftarrow$) If the coefficients of $F(X)$ are p-integral, then it can be coerced into $\mathbb{Z}[X]$ by replacing all denominators by their multiplicative inverse modulo p^e.

($\Rightarrow$) Since f is a polyfunction, there exists a representation $H(X) \in \mathbb{Z}[X]$ of degree less than $\mu(p^e)$. The polynomial $O(X) = H(X) - F(X)$ also has degree less than $\mu(p^e)$, and its evaluation function modulo p^e is zero. Writing

$$O(X) = \sum_{i=0}^{\mu(p^e)-1} \frac{a_i}{b_i} \cdot (X)_i,$$

where a_i/b_i is a fraction in simplest form, it suffices to prove $\nu_p(b_i) = 0$ for all i.

Assume on the contrary that $\nu_p(b_i) = \max_j\{\nu_p(b_j)\} = c > 0$, then $p^c \cdot O(X)$ can be coerced into a null polynomial modulo p^{c+e}. Since the degree of this null polynomial is strictly less than $\mu(p^e) \leqslant \mu(p^{c+e})$, it follows from Corollary 1 that

$$p^c \cdot \frac{a_i}{b_i} = 0 \pmod{p}.$$

From $\nu_p(b_i) = c$, it follows directly that $a_i = 0 \pmod{p}$. Hence both a_i and b_i are divisible by p, which contradicts the fact that a_i/b_i is in its simplest form. □

Remark 1. A polynomial $F(X) \in \mathbb{Q}[X]$ with non-p-integral coefficients can still represent a (poly)function $f \in \mathcal{F}_{p^e}$, for example if its degree is at least $\mu(p^e)$. However, it is not directly possible to evaluate such a function homomorphically.

Now we introduce a simple way to decide whether a given function $f \in \mathcal{F}_{p^e}$ is a polyfunction, relying on the reduced forward differences from Figure 2.

Theorem 3. *A function $f \in \mathcal{F}_{p^e}$ is a polyfunction if and only if the following two criteria are satisfied:*

1. *For all $i < \mu(p^e)$, we have $\nu_p(\alpha_{i,0}) \geqslant \nu_p(i!)$. Note that $\alpha_{i,0}$ are the diagonal elements of Figure 2.*
2. *All elements in the last row of Figure 2 are zero.*

Proof. ($\Leftarrow$) Consider the polynomial

$$F(X) = \sum_{i=0}^{p^e-1} \frac{\alpha_{i,0}}{i!} \cdot (X)_i \in \mathbb{Q}[X]. \tag{8}$$

Following Equation (3), this polynomial interpolates f in all data points. Now we are given that all elements in the last row of Figure 2 are zero, thus so are all values in the next row (which is not displayed). Hence $\alpha_{i,0} = 0$ for all $i \geqslant \mu(p^e)$. Therefore, we can terminate the summation of Equation (8) earlier and get

$$F(X) = \sum_{i=0}^{\mu(p^e)-1} \frac{\alpha_{i,0}}{i!} \cdot (X)_i \in \mathbb{Q}[X]. \tag{9}$$

Now it remains to prove that the coefficients of $F(X)$ are p-integral, and then the result follows immediately from Lemma 2. Considering Equation (9), this is trivial since we are given that $\nu_p(\alpha_{i,0}) \geqslant \nu_p(i!)$ for all $i < \mu(p^e)$.

($\Rightarrow$) If f is a polyfunction, it has a representation $F(X)$ of degree less than $\mu(p^e)$. Hence it follows from Equation (2) that $\overline{\Delta^{\mu(p^e)}} F(X)$ is zero in every point, which proves the second criterion of the theorem.

Consider again the polynomial of Equation (9). Following the same line of reasoning as in the first part of this proof, it is a representation of f modulo p^e. According to Lemma 2, we know that $F(X)$ must have p-integral coefficients, which implies $\nu_p(\alpha_{i,0}) \geqslant \nu_p(i!)$ for all $i < \mu(p^e)$. □

Interestingly, Equation (9) gives a polynomial representation $F(X)$ obtained by Newton interpolation restricted to $\{0, 1, \ldots, \mu(p^e) - 1\}$, i.e. only information about $f(i)$ for $i < \mu(p^e)$ has been used. Condition 1 of Theorem 3 can be interpreted as restricting the coefficients of $F(X)$ to p-integral values. Condition 2 is a consistency requirement: $F(a) = f(a) \pmod{p^e}$ for each $a \in \{\mu(p^e), \ldots, p^e - 1\}$. Finally, we note that also different interpolation methods could be used.

Corollary 3. *If f is a polyfunction, then Equation (9) gives a representation of the lowest degree.*

Proof. It was already proven that the polynomial $F(X)$ from Equation (9) can be coerced into a representation in $\mathbb{Z}[X]$, so it remains to show that its degree is minimal. Suppose that n is the largest integer such that $\alpha_{n,0} \neq 0$, and assume on the contrary that there exists a representation $H(X)$ whose degree is less than n. Then $O(X) = H(X) - F(X)$ is a null polynomial modulo p^e, with leading monomial $(\alpha_{n,0}/n!) \cdot X^n$. It follows from Corollary 1 that

$$\frac{\alpha_{n,0}}{n!} = 0 \pmod{p^{e-\nu_p(n!)}},$$

and thus $\alpha_{n,0} = 0 \pmod{p^e}$, leading to a contradiction. □

3.4 Bit and Digit Extraction Function

As an example, we apply the previously developed theory to the bit extraction function – a polyfunction that is useful in the part about FHE bootstrapping.

Example 2. Let $g_e \in \mathcal{F}_{2^e}$ be the bit extraction function defined as

$$g_e : \mathbb{Z}_{2^e} \to \mathbb{Z}_{2^e} : a \mapsto a \pmod 2,$$

where reduction modulo 2 is done in the set $\{0, 1\}$. Its forward differences are shown in Figure 3, which should be closely compared to Figure 2. The reduced forward differences are computed via reduction modulo 2^e. It can easily be verified in Table 1 that the diagonal elements $\alpha_{i,0} = (-2)^{i-1}$ satisfy $\nu_2(\alpha_{i,0}) \geqslant \nu_2(i!)$, and that all elements on the last row are congruent to zero modulo 2^e. Therefore, the bit extraction function is a polyfunction.

$$\begin{array}{cccccccccc}
0 & 1 & 0 & \cdots & 1 & \cdots & 0 & \cdots & 1 \\
 & 1 & -1 & \cdots & 1 & \cdots & -1 & \cdots & 1 \\
 & & -2 & \cdots & 2 & \cdots & -2 & \cdots & 2 \\
 & & & \ddots & \vdots & & \vdots & & \vdots \\
 & & & & (-2)^{i-1} & \cdots & -2^{i-1} & \cdots & 2^{i-1} \\
 & & & & & \ddots & \vdots & & \vdots \\
 & & & & & & (-2)^{\mu(2^e)-1} & \cdots & 2^{\mu(2^e)-1}
\end{array}$$

Fig. 3: Forward differences of the bit extraction function.

Following Corollary 3, the polynomial

$$G_e(X) = \sum_{i=1}^{e} \frac{(-2)^{i-1}}{i!} \cdot (X)_i \tag{10}$$

is a representation of g_e of the lowest degree. It follows that there does not exist a bit extraction polynomial of degree less than e. Finally, the complete set of representations is easily obtained as $G_e(X) + \mathcal{O}_{2^e}$.

More generally, we define the digit extraction function modulo p^e for any prime p from its balanced digit decomposition. Denote the balanced digits of $w \in \mathbb{Z}_{p^e}$ by $w_i \in \{-(p-1)/2, \ldots, (p-1)/2\}$ such that

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

then we define the map $g_e \in \mathcal{F}_{p^e}$ as

$$g_e : \mathbb{Z}_{p^e} \to \mathbb{Z}_{p^e} : w \mapsto w_0.$$

Analogously to the previous example, we can show that g_e is a polyfunction and obtain a representation of the lowest degree. In the general case, there does not exist a digit extraction polynomial of degree less than $(p-1)(e-1)+1$. The complete set of representations is obtained by adding $\mathcal{O}_{p^e}$.

3.5 Further Properties of Polyfunctions

Not every function is a polyfunction modulo p^e. For example, the function

$$f(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

is not a polyfunction for $e > 1$, because it is not *congruence preserving*. More specifically, all polyfunctions satisfy the following lemma.

Lemma 3 (Congruence preservation [7,9,16,17]). *Let f be a polyfunction modulo p^e, then for any $a \in \mathbb{Z}$, we have*

$$f(a + p^k) = f(a) \pmod{p^k}, \quad \forall k \leqslant e. \tag{11}$$

Proof. Let $F(X)$ be a representation of f. Since a polynomial is built from additions and multiplications only, we know that

$$F(a + p^k) = F(a) \pmod{p^k}.$$

Since $p^k \mid p^e$, we can directly replace F by f. This completes the proof. □

Congruence preservation is not a sufficient condition to be a polyfunction [4]. In Section 3.3 - Theorem 3, we derived a necessary and sufficient condition for a function to be a polyfunction based on reduced forward differences [17], which is consistent with the analytical characterization by Carlitz [7] and further leads to a representation of the lowest degree.

We can also give a sufficient but unnecessary condition for a function to be a polyfunction. A function that satisfies

$$f(a + p) = f(a) \pmod{p^e}, \quad \forall a \in \mathbb{Z}, \tag{12}$$

is said to have period p and is always a polyfunction. A representation can be derived as follows.

Lemma 4 (Adapted from [14]). *The polynomial $U(X) = 1 - X^{\varphi(p^e)}$ satisfies the following property modulo p^e:*

$$\forall a \in \mathbb{Z}\colon U(a) = \begin{cases} 1 & \text{if } p \mid a \\ 0 & \text{otherwise,} \end{cases}$$

where $\varphi(\cdot)$ is Euler's totient function.

A representation for a function f with period p is

$$F(X) = \sum_{k=0}^{p-1} f(k) \cdot U(X - k), \tag{13}$$

from which we can construct the set of complete representations $F(X) + \mathcal{O}_{p^e}$. A well-known example is the digit extraction function.

As shown by the next example, having period p is a sufficient, but not a necessary condition for a function to be a polyfunction.

Example 3. As computed in Example 1, there are $2^{8 \cdot 2^8} \approx 10^{617}$ functions in $\mathcal{F}_{2^8}$, while only $2^{50} \approx 10^{15}$ of them are polyfunctions. From these polyfunctions, only $2^{8 \cdot 2} \approx 10^5$ have period 2.

Even and Odd Polyfunctions. We construct a new lemma to find sparse representations of even and odd polyfunctions.

Lemma 5. *Let $f \in \mathcal{F}_{p^e}$ be an even (resp. odd) polyfunction, that is, $f(-a) = f(a) \pmod{p^e}$ (resp. $f(-a) = -f(a) \pmod{p^e}$) for $a \in \mathbb{Z}$. Moreover, assume that f has a degree-n representation. Then the following holds:*

- *If p is an odd prime, then f has a representation $F(X)$ of degree at most n, which contains only even (resp. odd) exponents.*
- *If $p = 2$, and we consider f modulo p^{e-1} instead of p^e, then it has a representation $F(X)$ of degree at most n, which contains only even (resp. odd) exponents.*

Proof. Consider a representation $H(X) \in \mathbb{Z}[X]$ of f that has degree equal to n. Due to the evenness (resp. oddness) of f, the polynomial $H'(X) = H(-X)$ (resp. $H'(X) = -H(-X)$) is an equivalent representation of f.

Now we consider the integer polynomial

$$F(X) = \frac{H(X) + H'(X)}{2}, \tag{14}$$

which contains only even (resp. odd) exponents and has degree at most n. By evaluating Equation (14) in any $a \in \mathbb{Z}$, we see that $F(a) = f(a) \pmod{p^e}$ for an odd prime p, and $F(a) = f(a) \pmod{p^{e-1}}$ for $p = 2$. Hence $F(X)$ is also a representation of f, and it can easily be checked that it contains only even (resp. odd) exponents. □

4 Faster Bootstrapping for BGV and BFV

This section explains our improved bootstrapping techniques for BGV and BFV, leveraging the observations from the first part of the paper. Both general and thin bootstrapping involve two important components: the linear transformations and digit removal. We do not propose adaptations to the linear transformations, and

leave them unchanged in the implementation. Our improvements are inside the digit removal step, and follow from the polyfunctions theory. Since digit removal is 3× to 50× more expensive than the linear transformations [15], any speedup leads to an almost equal effect in the entire bootstrapping procedure.

4.1 Cost Model

Amdahl's law [2] states that the speedup gained by optimizing a single part of an algorithm is limited to the fraction of time that the improved part is used. In order to accelerate digit removal, we must therefore concentrate on the slowest and most commonly used FHE operations. The true bottleneck of digit removal is *non-scalar multiplication*, i.e. multiplication of two ciphertexts. For an example parameter set with ring dimension $N = 2^{16}$, non-scalar multiplication in `HElib` is 7× more expensive than its scalar counterpart.

An approach that follows our cost model is the baby-step/giant-step algorithm for evaluating a set of polynomials with scalar coefficients in a common non-scalar point [12,20]. It can asymptotically evaluate m degree-n polynomials with $2\sqrt{mn}$ non-scalar multiplications. Therefore, our implementation uses this algorithm for polynomial evaluation.

Although not the focus of this paper, digit removal is also costly in terms of multiplicative depth (which is by definition the maximal number of multiplications encountered in each possible input-output path). Our approach accelerates bootstrapping without significantly affecting the multiplicative depth of digit removal. This is achieved by exclusive use of low-degree polynomials.

4.2 Digit Removal Algorithm

The digit removal procedure removes the v least significant digits of its input $w \in \mathbb{Z}_{p^e}$ for a given prime number p and $v < e$. Formally, for odd p, denote the balanced digits of $w \in \mathbb{Z}_{p^e}$ by $w_i \in \{-(p-1)/2, \ldots, (p-1)/2\}$ such that

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

Digit removal is then defined as the map

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

In other words, it consecutively scales and rounds the input. This is necessary in bootstrapping to remove the noise. To evaluate the procedure homomorphically, it is written as a series of polynomial evaluations and divisions.

Note that the balanced digit representation only exists for odd prime numbers. If $p = 2$, we need to consider the digits in $\{0, 1\}$, which causes the output of digit removal to be $\lfloor w/p^v \rfloor$. However, bootstrapping requires a rounding operation instead of flooring. This can be fixed by applying the simple equality $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ just before digit removal.

Existing Digit Removal Algorithms. Digit removal uses the following notation: we write $w_{i,j}$ for *any* integer of which the least significant digit is w_i, and the next j least significant digits are all zeros. Formally, this means that $w_{i,j} = w_i \pmod{p^{j+1}}$. Moreover, we require two well-known polynomials:

- The *lifting polynomial* $F_e(X) \in \mathbb{Z}[X]$ satisfies $F_e(w_{i,j}) = w_{i,j+1}$ for $j \leqslant e$. In other words, it allows us to compute a valid $w_{i,j+1}$ from any given $w_{i,j}$ by zeroing one extra digit.
- The *digit extraction polynomial* $G_e(X) \in \mathbb{Z}[X]$ satisfies $G_e(w_{i,0}) = w_{i,e-1}$, which allows us to compute a valid $w_{i,e-1}$ from any given $w_{i,0}$ by zeroing $e-1$ extra digits. In other words, it is a representation of the digit extraction function g_e introduced in Section 3.3.

It can be proven that the above polynomials always exist [8]. Their degrees are respectively p and $(e-1)(p-1)+1$.

The high-level idea of digit removal is to use the lifting polynomial and/or digit extraction polynomial to extract the least significant digit of the input w. The result is then subtracted from w and divided by p, and this is repeated until enough digits are removed. A suitable choice of lifting and digit extraction polynomials ensures a low multiplicative depth of the resulting procedure. Digit removal is visualized in the trapezoid of Figure 4 for an example parameter set of $e = 5$ and $v = 3$. The procedure works as follows:

- We start from $w_{0,0} = w$ in the first row, and then compute the numbers on its right via a series of polynomial evaluations. The choice of polynomials depends on the chosen algorithm, and is explained later.
- In the second row, we first compute $w_{1,0} = (w - w_{0,1})/p$ and then repeat the same procedure from the first row.
- In the last row, we similarly compute $w_{2,0} = ((w - w_{0,2})/p - w_{1,1})/p$ and again repeat the same procedure from the first and second row.
- The result is obtained as $(((w - w_{0,4})/p - w_{1,3})/p) - w_{2,2}/p$. This is omitted from the figure.

In summary, the first digit of each row is computed by subtracting the digits on the same diagonal and dividing by p. All other digits are obtained via a series of polynomial evaluations, starting from the first digit in its row. Finally, the result is obtained by subtracting the last digit of each row from the input and dividing by p.

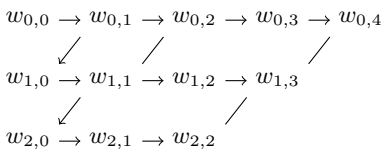

Fig. 4: Visualization of digit removal for $e = 5$ and $v = 3$.

Until now, we have only specified operations *between rows*, which are identical for all methods that we will discuss (including our own). Existing digit removal procedures differ in how they compute digits *within the same row*. Two different methods have been proposed for this: the first one is from Halevi and Shoup [15], and the second one from Chen and Han [8].

Halevi/Shoup Digit Removal. This procedure computes each number in the trapezoid (except for the first one in each row) by applying the lifting polynomial to the number on its left. In other words, we use the identity $w_{i,j+1} = F_e(w_{i,j})$. The cost is dominated by $ev - v(v+1)/2$ evaluations of the lifting polynomial. The degree of this procedure is roughly p^{e-1}.

Chen/Han Digit Removal. This procedure computes the last number of each row by applying the digit extraction polynomial to the first number of the same row. In other words, we use the identity $w_{i,j} = G_{j+1}(w_{i,0})$. All other digits are computed identical to the Halevi/Shoup procedure; but note that some digits are not used and can therefore be omitted. In the example of Figure 4, we do not need to compute $w_{0,3}$, $w_{1,2}$ and $w_{2,1}$.

The cost of Chen/Han digit removal is dominated by $v(v-1)/2$ evaluations of the lifting polynomial and v evaluations of the digit removal polynomial. However, its main advantage is in degree, which is roughly equal to rp^v with $r = e - v$. During bootstrapping, the parameter r represents the precision of the plaintext space, i.e. plaintexts are computed modulo p^r. As such, the Chen/Han procedures has asymptotically lower degree than Halevi/Shoup for high-precision plaintext spaces.

4.3 Faster Digit Removal

In the following sections, we apply five changes to the original Chen/Han digit removal procedure. The first adaptation relates to digit removal itself, and is a better method to evaluate the polynomials of each row. The other improvements follow from polyfunctions theory.

Adapted Row Computation. As already mentioned earlier, digit removal can be analyzed row per row, where Halevi and Shoup take a different approach than Chen and Han. In contrast to both these versions, we propose a method that uses the digit extraction polynomial exclusively, without relying on the lifting polynomial. Specifically, we compute each element of the trapezoid by applying a suitable digit extraction polynomial to the first element in the same row. This has two advantages: firstly, all polynomials can be evaluated *simultaneously* using the baby-step/giant-step technique. Due to the $2\sqrt{mn}$ complexity, this leads to a performance benefit over evaluating all polynomials separately. Secondly, this method works well in conjunction with our next optimization (finding a more efficient representation of the digit extraction polynomial).

In instantiating our method, we need to avoid depth increase of the resulting procedure as much as possible. In particular, we have to be careful with the

path along the evaluated circuit of largest depth, referred to as the *critical path*. Any depth increase in the critical path causes a corresponding depth increase in the entire procedure. The critical path of Chen/Han digit removal is depicted in Figure 5. It runs via the vertical dimension first, because the depth grows linearly there and logarithmically in the horizontal dimension.

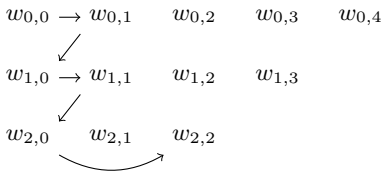

Fig. 5: Critical path of digit removal for $e = 5$ and $v = 3$.

In a first attempt, we can compute each digit as $w_{i,j} \leftarrow G_{j+1}(w_{i,0})$. In some cases, however, we can do better by reusing computed elements. In particular, we can set $w_{i,k} \leftarrow w_{i,j}$ for $k < j$ without affecting correctness; however, also this is not always desirable because it can lead to a depth increase of the digit removal procedure. For example, it is never beneficial to take $w_{i,1} \leftarrow w_{i,j}$ in terms of noise growth, because $w_{i,1}$ lies on the critical path. Our implementation takes the heuristic approach of computing $w_{i,j} \leftarrow G_{j+1}(w_{i,0})$ for each value of $j + 1$ that is a power of 2, and setting $w_{i,j} \leftarrow w_{i,j+1}$ otherwise. This heuristic does not increase the multiplicative depth compared to Chen/Han digit removal.

Even and Odd Functions. The digit extraction function for $p = 2$ is an even function. Following Lemma 5, we can find a representation of degree $e + 1$ or less that contains only even exponents. Specifically, we write the digit extraction polynomial as $G_e(X) = F(X^2)$ for some polynomial $F(X)$ of degree $\lfloor (e + 1)/2 \rfloor$. Such polynomials can be evaluated more efficiently than regular ones by first computing X^2 before applying a standard baby-step/giant-step method. This requires asymptotically $\sqrt{2mn}$ non-scalar multiplications for evaluating m polynomials of degree n.

Similarly to the case above, the digit extraction function for an odd prime p is an odd function. Following Lemma 5, we can find a representation of degree $(e - 1)(p - 1) + 1$ that contains only odd exponents. Specifically, we write the digit extraction polynomial as $G_e(X) = X \cdot F(X^2)$ for some polynomial $F(X)$ of degree $(e-1)(p-1)/2$. Such polynomials can be evaluated more efficiently than regular ones, using one the methods of Lee et al. [18]. Their first method omits unused powers of X in the baby-step, and can be evaluated with optimal multiplicative depth. Their second method first evaluates $F(X^2)$ using the strategy from above, and then multiplies by X. This increases the depth by at most one. Both methods require asymptotically $\sqrt{2mn}$ non-scalar multiplications for eval-

uating m polynomials of degree n. All experiments in Section 5 are conducted with the first variant.

Function Composition. We propose a new method to obtain the digit extraction function modulo p^e by decomposing it as $g_e = g_{e,e'} \circ g_{e'}$ for some $e' < e$. In our method, the relevant domain of $g_{e,e'}$ is therefore no longer $\mathbb{Z}$, but rather the range of $g_{e'}$. Our analysis starts from the following definitions.

Definition 7. *Let $f \in \mathcal{F}_{p^e}$ be a function from $\mathbb{Z}_{p^e}$ to itself. If there exists a polynomial $F(X) \in \mathbb{Z}[X]$ that satisfies $F(a) = f(a) \pmod{p^e}$ for all $a \in S \subseteq \mathbb{Z}$, then f is a* polyfunction modulo p^e over S *and $F(X)$ is a* representation *of f.*

Definition 8. *An element $O(X) \in \mathbb{Z}[X]$ is called a* null polynomial modulo p^e over $S \subseteq \mathbb{Z}$ *if the function $f \in \mathcal{F}_{p^e}$ that it represents maps every element from S to zero. In other words, we have that $O(a) = 0 \pmod{p^e}$ for all $a \in S$.*

The inner function $g_{e'}$ can directly be represented as a polynomial in the even or odd representation. For the outer function $g_{e,e'}$, we can use the adapted definitions from above, where we define the set

$$S = \left\{ k + i \cdot p^{e'} : \ -(p-1)/2 \leqslant k \leqslant (p-1)/2 \text{ and } i \in \mathbb{Z} \right\}. \tag{15}$$

This coincides with the range of $g_{e'}$. For $p = 2$, we slightly need to change the definition of S and allow $0 \leqslant k \leqslant 1$.

Since digit extraction is an idempotent operation, one possible representation of $g_{e,e'}$ is $G_e(X)$. But the domain of $g_{e,e'}$ is restricted to S, so we can find other representations by adding null polynomials that satisfy Definition 8. Therefore, our problem reduces to studying null polynomials over S, which we can construct as follows. Consider

$$H_j(X) = \begin{cases} (X)_j & \text{if } p = 2 \\ \left(X + \frac{p-1}{2}\right)_j & \text{if } p \text{ is an odd prime.} \end{cases} \tag{16}$$

To ease notation, we also write $H(X) = H_p(X)$. Moreover, let

$$(X)_{i,j} = \left(\prod_{k=0}^{i-1} H(X - k \cdot p^{e'}) \right) H_j(X - i \cdot p^{e'}) \tag{17}$$

for $0 \leqslant j < p$. Then we can adapt Theorem 1 as follows.

Theorem 4. *A polynomial $O(X) \in \mathbb{Z}[X]$ is a null polynomial modulo p^e over the set S from Equation (15) of degree at most n if and only if there exist $a_{i,j} \in \mathbb{Z}$ such that*

$$O(X) = \sum_{0 \leqslant d(i,j) \leqslant n} a_{i,j} \cdot O_{i,j}(X), \quad \textit{with} \quad O_{i,j}(X) = p^{\max(e - i \cdot e' - \nu_p(i!), 0)} \cdot (X)_{i,j}.$$

In this equation, the function $d(i,j) = p \cdot i + j$ denotes the degree of $O_{i,j}(X)$. It is also implicitly assumed that $0 \leqslant j < p$.

Proof. ($\Leftarrow$) Evaluating Equation (17) at any $a \in S$ gives

$$(X)_{i,j}\,|_{X=a} = \left(\prod_{k=0}^{i-1} H(a - k \cdot p^{e'})\right) H_j(a - i \cdot p^{e'}). \tag{18}$$

From the definition of $H(X)$ in Equation (16) and the restriction to S, it follows that $H(a)$ is divisible by $p^{e'}$. In fact, exactly one factor of $H(X)$ will be divisible by $p^{e'}$ when evaluated at $X = a$. Let $X - q$ be this linear factor where $0 \leqslant q \leqslant 1$ (if $p = 2$) or $-(p-1)/2 \leqslant q \leqslant (p-1)/2$ (if p is odd), then we can set $a - q = \alpha \cdot p^{e'}$ for some α. Equation (18) is then divisible by

$$\prod_{k=0}^{i-1} \left(a - q - k \cdot p^{e'}\right) = \prod_{k=0}^{i-1} \left(\alpha \cdot p^{e'} - k \cdot p^{e'}\right) = p^{i \cdot e'} \cdot (X)_i\,|_{X=\alpha}.$$

As already pointed out in Section 2.3, the evaluation of $(X)_i$ at any integer is divisible by $p^{\nu_p(i!)}$. Hence our result is divisible by $p^{i \cdot e' + \nu_p(i!)}$, and it follows that $O_{i,j}(X)|_{X=a}$ is divisible by $p^{\max(e, i \cdot e' + \nu_p(i!))} \geqslant p^e$. Any $\mathbb{Z}$-linear combination of these $O_{i,j}(X)$ is thus a null polynomial modulo p^e over S.

($\Rightarrow$) We prove the following assertion for $0 \leqslant m \leqslant n + 1$ by applying induction on m:

$$O(X) = \sum_{m \leqslant d(i,j) \leqslant n} b_{i,j} \cdot (X)_{i,j} + \sum_{0 \leqslant d(i,j) < m} a_{i,j} \cdot O_{i,j}(X), \tag{19}$$

for some $a_{i,j}, b_{i,j} \in \mathbb{Z}$.

The base case $m = 0$ is trivial since the second sum is empty, and the first sum amounts to writing a polynomial in the basis given by $(X)_{i,j}$. It is therefore possible to find appropriate constants $b_{i,j}$ that satisfy Equation (19).

Now suppose that Equation (19) was established for some $m < n + 1$, that is

$$O(X) = b_{i',j'} \cdot (X)_{i',j'} + \sum_{m < d(i,j) \leqslant n} b_{i,j} \cdot (X)_{i,j} + \sum_{0 \leqslant d(i,j) < m} a_{i,j} \cdot O_{i,j}(X),$$

with $d(i', j') = m$. Evaluating both sides at $X = a$ with $a = i' \cdot p^{e'} + j'$ (if $p = 2$) or $a = i' \cdot p^{e'} + j' - (p-1)/2$ (if p is odd) gives

$$0 = O(a) = b_{i',j'} \cdot \prod_{k=0}^{i'-1} (X)_p\,|_{X=(i'-k) \cdot p^{e'} + j'} \cdot (j')! \pmod{p^e}.$$

Taking the p-adic valuation of the right-hand side and following a similar line of reasoning as in the first part of this proof, we get

$$\nu_p\left(b_{i',j'} \cdot \prod_{k=0}^{i'-1} (X)_p\,|_{X=(i'-k) \cdot p^{e'} + j'} \cdot (j')!\right) = \nu_p(b_{i',j'}) + i' \cdot e' + \nu_p((i')!) \geqslant e.$$

The constants $b_{i',j'}$ are integers, so $\nu_p(b_{i',j'}) \geqslant \max(e - i' \cdot e' - \nu_p((i')!), 0)$. We can therefore write $b_{i',j'} = a_{i',j'} \cdot p^{\max(e-i'\cdot e'-\nu_p((i')!),0)}$ for some $a_{i',j'} \in \mathbb{Z}$, which results in

$$O(X) = \sum_{m<d(i,j)\leqslant n} b_{i,j} \cdot (X)_{i,j} + \sum_{0\leqslant d(i,j)\leqslant m} a_{i,j} \cdot O_{i,j}(X).$$

This expression replaces m by $m+1$ in Equation (19) and thereby completes the induction. The final result follows by setting $m = n + 1$ in Equation (19). □

To study the degree of null polynomials restricted to the set S, we consider an adapted variant of the Smarandache function that takes two inputs:

$$\mu_p(e, e') = \min\{i \in \mathbb{N} : e' \cdot i + \nu_p(i!) \geqslant e\}.$$

Then it is clear that

$$O_{\mu_p(e,e'),0}(X) = \prod_{k=0}^{\mu_p(e,e')-1} H(X - k \cdot p^{e'}) \tag{20}$$

is a monic null polynomial of degree $p \cdot \mu_p(e, e') \approx p \cdot \lceil e/e' \rceil$.

Now we have all ingredients available to find a better representation of $g_{e,e'}$. Starting from $G_e(X)$, we can apply Euclidean division and reduce it modulo the null polynomial of Equation (20). This results in a representation $G_{e,e'}(X)$ that has degree strictly less than $p \cdot \lceil e/e' \rceil$. This can be much smaller than the degree of $G_e(X)$, which is equal to $(e - 1)(p - 1) + 1$.

For odd primes p, the function $g_{e,e'}$ is odd and we can directly choose $G_{e,e'}(X)$ with only odd-exponent terms. However, if $p = 2$ then $-S \nsubseteq S$, hence $g_{e,e'}$ is not defined for all inputs from $-S$. The function is therefore not even, and we cannot directly choose $G_{e,e'}(X)$ with only even-exponent terms. One possible solution is to allow $-1 \leqslant k \leqslant 1$ in Equation (15) instead of $0 \leqslant k \leqslant 1$. However, this increases the degree of the polynomial from Equation (20) by 50%, hence also the degree of the resulting polynomial representation. We did not incorporate this solution in our implementation.

Finally, we note that the set of null polynomials modulo p^e over S of degree bound n still forms a p^e-ary lattice. This lattice is given by

$$\left\{ \sum_{0\leqslant d(i,j)\leqslant n} a_{i,j} \cdot O_{i,j}(X) \mid a_{i,j} \in \mathbb{Z} \right\} \subseteq \mathbb{P}_n,$$

where the basis vectors are $O_{i,j}(X)$.

Asymptotic Complexities. We now analyze the asymptotic depth and time complexities of our composite approach. Specifically, its depth is bounded by

$$\lceil \log_2 ((p-1) \cdot (e'-1) + 1) \rceil + \left\lceil \log_2 \left(p \cdot \left\lceil \frac{e}{e'} \right\rceil \right) \right\rceil \approx \lceil \log_2 e \rceil + 2 \cdot \lceil \log_2 p \rceil,$$

counting only non-scalar multiplications. The first term comes from $g_{e'}$ and the second one from $g_{e,e'}$. When compared to the regular $G_e(X)$, there is a depth increase of $\lceil \log_2 p \rceil$. Note that we can also apply function composition multiple times in a row, and then the depth will increase with $\lceil \log_2 p \rceil$ per stage. In terms of scalar multiplications, there is a depth increase of 1 for each stage of function composition. For the sake of noise control, our approach favors a small number of stages and a low value of p.

Performance-wise, we make a difference between scalar and non-scalar multiplications. The baby-step/giant-step technique can asymptotically evaluate a polynomial of degree n with $2\sqrt{n}$ non-scalar and n scalar multiplications. If the polynomial is even or odd, these numbers reduce to respectively $\sqrt{2n}$ and $n/2$. As such, we have the following time complexities for the digit extraction function:

- For $p = 2$, the original method can evaluate the digit extraction polynomial asymptotically with $\sqrt{2e}$ non-scalar and $e/2$ scalar multiplications. Our composite approach reduces this to respectively $\sqrt{2e'}+2\sqrt{2e/e'}$ and $e'/2+2e/e'$. The number of non-scalar multiplications is minimal if $e' = 2\sqrt{e}$, which gives $4\sqrt[4]{e}$ non-scalar and $2\sqrt{e}$ scalar multiplications. Since $p = 2$, this analysis assumes that $G_{e,e'}(X)$ has both even- and odd-exponent terms.
- For larger values of p, the original method can evaluate the digit extraction polynomial asymptotically with $\sqrt{2pe}$ non-scalar and $pe/2$ scalar multiplications. Our composite approach reduces this to respectively $\sqrt{2p}(\sqrt{e'}+\sqrt{e/e'})$ and $(p/2) \cdot (e' + e/e')$. The number of non-scalar multiplications is minimal if $e' = \sqrt{e}$, which gives $2\sqrt{2p}\sqrt[4]{e}$ non-scalar and $p\sqrt{e}$ scalar multiplications. Since $p \neq 2$, this analysis assumes that $G_{e,e'}(X)$ has only odd-exponent terms. Moreover, the degree of $G_e(X)$ is approximated as pe.

In conclusion, our method reduces the number of non-scalar multiplications from $\mathcal{O}(\sqrt{pe})$ to $\mathcal{O}(\sqrt{p}\sqrt[4]{e})$ asymptotically. The number of scalar multiplications are reduced from $\mathcal{O}(pe)$ to $\mathcal{O}(p\sqrt{e})$.

Table 3 shows the number of operations to evaluate digit extraction, comparing the Halevi/Shoup and Chen/Han method to our approach. The even and odd entries represent the standard version (without function composition). The tuples represent the indices (e, e', e'') of the digit extraction function, where e'' is the index of the innermost function and e is the index of the outermost function. All tuples are chosen to minimize the number of non-scalar multiplications. It is clear from the table that our composite method works especially well for low p and high e, where the performance benefits can be fully exploited without a significant depth increase. On the other hand, it turns out that even for large values of e (up to 256), splitting in more than 2 stages does not (much) increase performance anymore.

Lattices. Another method to find better polynomial representations is via lattice problems. Given a polynomial $F(X)$ and the null lattice, we can solve the closest vector problem to find a null polynomial $O(X)$ that lies closest to $F(X)$. The representation $F(X) - O(X)$ is then equivalent to the original one, but it has smaller coefficients. This leads to less noise growth in FHE ciphertexts.

Table 3: Non-scalar depth and operation count for the digit extraction function.

p	e	method	depth	#(non-scalar mults)	#(scalar mults)
2	64	H/S	63	63	0
		C/H	6	16	64
		Our even	6	12	32
		(64, 16)	7	9	15
	256	H/S	255	255	0
		C/H	8	33	256
		Our even	8	25	128
		(256, 32)	9	15	31
		(256, 67, 16)	10	13	22
3	64	H/S	126	126	0
		C/H	7	24	127
		Our odd	7	20	64
		(64, 16)	9	16	22
		(64, 25, 8)	10	15	24
	256	H/S	510	510	0
		C/H	9	49	511
		Our odd	9	38	256
		(256, 24)	11	23	40
		(256, 92, 8)	12	21	58

This lattice trick can also be combined with all earlier described techniques. For even or odd functions, we can start from a lattice that only contains even or odd null polynomials. These can be found via simple linear algebra on the original null lattice. For the function composition approach, we can start from the null lattice restricted to the set S as defined earlier.

Example 4. The advantage of using lattices is demonstrated on the bit extraction polynomial. Our method was able to find the following representations for $p = 2$:

- Bit extraction modulo 2^8 can be done with $G_8(X) = 13X^8 - 12X^6$.
- Starting from the result modulo 2^8, bit extraction modulo 2^{25} can be done with $G_{25,8}(X) = 6X^5 - 15X^4 + 10X^3$.

Both polynomials have remarkably small coefficients, since they are defined modulo 2^8 and 2^{25} respectively.

Multivariate Approach. We considered one more strategy to compute better polynomial representations based on multivariate equations. The idea is to write out consecutive digit extraction polynomials in a pattern that minimizes the non-scalar multiplications. This gives a system of non-linear equations, which can be solved for the coefficients of the digit extraction polynomial. Although this strategy does not generalize to higher parameters, we were able to find bit extraction polynomials for $e \leqslant 16$ that can be evaluated with $\lceil \log_2 e \rceil$ non-scalar multiplications, which is provably minimal. These instances are listed in Table 4.

Table 4: Recursive evaluation of the bit extraction polynomial.

e	$G_e(X)$
2	X^2
4	$G_2(X)^2$
8	$112 \cdot G_2(X) + (94 \cdot G_2(X) + 121 \cdot G_4(X))^2$
16	$11136 \cdot G_4(X) + (28504 \cdot G_2(X) + 8968 \cdot G_4(X) - G_8(X)) \cdot (15364 \cdot G_4(X) + 14115 \cdot G_8(X))$

5 Implementation and Performance

We implemented our new digit removal procedure for the BGV scheme in `HElib`. For two reasons, we did not implement it for the BFV scheme: firstly, there is no software library that supports BFV bootstrapping; secondly, BGV and BFV are known to be equivalent in terms of bootstrapping, and only differ in some minor implementation details. Therefore, any performance discrepancy would reflect the underlying arithmetic operations and not our improvements.

We give experimental results for general bootstrapping in Table 5 and for thin bootstrapping in Table 6. The tables show capacity (number of bits in the noise) and execution time. The factorization of the parameter m determines the complexity of the linear maps (explained in [15]), but is not relevant for digit removal. The regular plaintext modulus is p^r, which is augmented to p^e during bootstrapping. The function composition method was not used for these tables, since its effect is thoroughly analyzed in Section 5.1. All experiments were run on a single-threaded Intel® Core™ i7-6700HQ CPU with 8 GB memory and Ubuntu 22.04.1 LTS installed.

The "improvement" in the last row of Tables 5 and 6 was computed in the amortized sense, i.e. as the ratio

$$\text{improvement} = \frac{\text{old execution time}}{\text{new execution time}} \cdot \frac{\text{new remaining capacity}}{\text{old remaining capacity}}.$$

We achieve a significant improvement for all tested parameter sets, ranging from 1.3× to 2.6×. The speedups are higher for general bootstrapping than for thin bootstrapping. The reason is that the general version requires multiple digit removals, whereas the thin version requires only one. In terms of noise capacity, the advantage also tends to be in our direction. This is likely a consequence of two facts: we replaced the lifting polynomial by the digit extraction polynomial of lower degree; and the coefficients of our polynomials are smaller due to the lattice trick.

5.1 Function Composition

Our implementation also includes the function composition approach, which is asymptotically cheaper for high-precision plaintext spaces (i.e. large values of r

and e). We demonstrate its benefit in Table 7 for plaintext moduli up to 2^{59}. For technical reasons (not inherent to the BGV scheme), `HElib` does not support more than 59 bits of precision, so we could not test with higher values than this. Furthermore, bootstrapping is only supported for $p^e < 2^{30}$ [15], so it is impossible to run bootstrapping with the parameter sets from Table 7. We therefore show the results for digit extraction only. Finally, note that the parameters e and e' have the same meaning as in Section 4.3.

Table 5: General bootstrapping in `HElib` (original/ours).

Cyclotomic index m		$127 \cdot 337$	$101 \cdot 451$	$43 \cdot 757$
Number of slots		2016	1000	2268
Params (p, r, e)		$(2, 8, 15)$	$(17, 4, 6)$	$(127, 2, 4)$
Security level (bits)		81	78	66
Number of digit removals		21	40	14
Capacity (bits)	Initial	1151	1136	1134
	Linear maps	100	147	140
	Digit extract	307/298	541/514	671/712
	Remaining	744/753	448/475	323/282
Execution time (sec)	Linear maps	134	150	290
	Digit extract	2014/743	2665/1879	1407/863
	Total	2248/877	2815/2029	1697/1153
Improvement		2.6×	1.5×	1.3×

Table 6: Thin bootstrapping in `HElib` (original/ours).

Cyclotomic index m		$127 \cdot 337$	$101 \cdot 451$	$43 \cdot 757$
Number of slots		2016	1000	2268
Params (p, r, e)		$(2, 8, 15)$	$(17, 4, 6)$	$(127, 2, 4)$
Security level (bits)		81	78	66
Capacity (bits)	Initial	1151	1136	1134
	Linear maps	137	174	164
	Digit extract	267/260	445/435	604/640
	Remaining	747/754	517/527	366/330
Execution time (sec)	Linear maps	35	32	31
	Digit extract	105/35	65/46	101/64
	Total	140/70	97/78	132/95
Improvement		2.0×	1.3×	1.3×

The table has four values per column: the original built-in implementation; our standard method without function composition; our method with partial function composition; and our method with full function composition. The third

value of each column (partial function composition) is generated by applying function composition to each row of Figure 5, except for the last one. The motivation for this is as follows. Since the bottom right element of Figure 5 lies on the critical path, it determines the multiplicative depth of digit removal. Turning this argument around, we can reduce the depth by not applying function composition in the last row. In other words, there is a depth-efficiency trade-off, where function composition favors efficiency and the standard method favors depth.

The "speedup" in the last row of Table 7 was computed as the ratio between the original approach and our three methods:

$$\text{speedup} = \frac{\text{old execution time}}{\text{new execution time}}.$$

This cost measure ignores the remaining noise capacity, because we cannot run the full bootstrapping procedure and therefore don't have this number available. Again, we achieve major speedups compared to `HElib`'s built-in digit removal, ranging from 1.6× to 2.8×. Since digit removal is the main bottleneck, bootstrapping would exhibit almost identical speedups.

Table 7: High-precision digit removal in `HElib` (original/our standard method/partial function composition/full function composition).

Cyclotomic index m		42799	63973
Number of slots		2016	2592
Params (p, r, e, e')		$(2, 51, 59, 16)$	$(3, 32, 37, 6)$
Security level (bits)		80	77
Capacity (bits)	Initial	1137	1335
	Digit extract	1049/991/970/1006	1142/1047/1103/1170
Execution time (sec)		180/100/67/64	191/151/124/119
Speedup		1.8×/2.7×/2.8×	1.3×/1.5×/1.6×

6 Conclusion

Although polynomial functions over rings are commonly used in cryptography, their properties are currently not well understood. This paper contributed to the analysis of such polyfunctions, including existence, computation and equivalence of polynomial representations, among other things.

Our theory is directly applicable to FHE bootstrapping: we found sparse representations (either even or odd) for the digit extraction function, which is the bottleneck in bootstrapping; we also proposed a new method to decompose digit extraction into multiple stages, each of which can be evaluated with a polynomial of low degree. Altogether, we observed speedups of up to 2.6× for bootstrapping and up to 2.8× for digit removal, including our function composition approach.

Finding the optimal way to evaluate the polynomials during bootstrapping, taking into account both noise growth and execution time, remains an interesting open problem.

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, and in part by the Research Council KU Leuven grant C14/18/067. Robin Geelen is funded in part by Research Foundation – Flanders (FWO) under a PhD Fellowship fundamental research (project number 1162123N).

References

1. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Annual Cryptology Conference. pp. 1–20. Springer (2013)
2. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, spring joint computer conference. pp. 483–485 (1967)
3. Araki, T., Barak, A., Furukawa, J., Keller, M., Lindell, Y., Ohara, K., Tsuchida, H.: Generalizing the spdz compiler for other protocols. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 880–895 (2018)
4. Bhargava, M.: P-orderings and polynomial functions on arbitrary subsets of dedekind rings. Journal für die reine und angewandte Mathematik (Crelles Journal) **1997**(490-491), 101–128 (1997), https://doi.org/10.1515/crll.1997.490.101
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science. pp. 309–325. Association for Computing Machinery (Jan 2012). https://doi.org/10.1145/2090236.2090262
7. Carlitz, L.: Functions and polynomials (mod p^n). Acta Arithmetica **9**(1), 67–78 (1964), http://eudml.org/doc/207463
8. Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 315–337. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_12
9. Chen, Z.: On polynomial functions from $\mathbb{Z}_n$ to $\mathbb{Z}_m$. Discrete Mathematics **137**(1-3), 137–145 (1995)
10. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), http://eprint.iacr.org/2012/144

11. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
12. 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)
13. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography – PKC 2012. pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
14. Guha, A., Dukkipati, A.: An algorithmic characterization of polynomial functions over $\mathbb{Z}_{p^n}$. Algorithmica **71**(1), 201–218 (2015)
15. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(1), 1–44 (2021)
16. Keller, G., Olson, F.R.: Counting polynomial functions (mod p^n). Duke Mathematical Journal **35**(4), 835–838 (1968)
17. Kempner, A.J.: Polynomials and their residue systems. Transactions of the American Mathematical Society **22**(2), 240–288 (1921)
18. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2020/552 (2020), https://eprint.iacr.org/archive/2020/552/20200803:084202
19. Li, S.: Null polynomials modulo m. arXiv preprint math/0510217 (2005)
20. 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)
21. Singmaster, D.: On polynomial functions (mod m). Journal of Number Theory **6**(5), 345–352 (1974)
22. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**(1), 57–81 (2014)
23. Specker, E., Hungerbühler, N., Wasem, M.: The ring of polyfunctions over $\mathbb{Z}/n\mathbb{Z}$ (2021)