Faster Polynomial Evaluations for SIMD FHEs and Application to BGV in HElib

Jiachen Zhao[1,2,3], Jiang Zhang[1(✉)], Binwu Xiang[4(✉)], Songyu Wu[2,3], Yi Deng[5(✉)], and Dengguo Feng[1(✉)]

[1] State Key Laboratory of Cryptology, Beijing, China
zhaojiachen@iie.ac.cn, zhangj@sklc.org, fengdg@263.net
[2] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[4] East China Normal University, Shanghai, China
bwxiang@sc.ecnu.edu.cn
[5] School of Cryptology, Xidian University, Xi'an, China
deng@iie.ac.cn

Abstract. The cost of homomorphic multiplications for existing FHEs to evaluate a degree-D polynomial $f(x)$ at some point x is very expensive. When x is encoded in a plaintext slot having a power-of-two degree $d = 2^\ell$ and $D \leq d$, one can efficiently evaluate $f(x)$ with $O(\log d)$ multiplications using the heuristic algorithms of Okada et al. (ASIACRYPT 2023). However, neither $d = 2^\ell$ nor $D \leq d$ is satisfied for most practical FHE parameters, and the Paterson–Stockmeyer (P-S) method with $O(\sqrt{D})$ multiplications remains the state-of-the-art for $d \neq 2^\ell$ or $D > d$.

In this paper, we first present a polynomial evaluation algorithm with $O(\log d)$ multiplications for any non-power-of-two d and $D \leq d$, which achieves the same asymptotic complexity as that of Okada et al. Then, we gave a polynomial evaluation algorithm with $O(\sqrt{D/d})$ multiplications for plaintext modulus $p > 2$ and $d < D \leq d \log p$, which beats the P-S method by a factor of $\sqrt{d}$ and essentially achieves logarithmic multiplication complexity when $D \leq d \cdot \min(\log^2 D, \log p)$. As a major application, we implement our algorithms in experiment to evaluate the digit extraction polynomials of the BGV bootstrapping with parameter d ranging from 14 to 45 in HElib[6], and obtain a $1.22 - 2.16\times$ speedup over the recent work of Ma et al. (EUROCRYPT 2024).

Keywords: FHE · BGV · Polynomial Evaluation · Bootstrapping

[6] The code is public available at https://github.com/Nobody673/artifact-helib/tree/main.

1 Introduction

Fully homomorphic encryption (FHE) is a powerful cryptographic primitive, which allows one to perform computation over encrypted data without learning anything about the data. Since the first FHE construction due to Gentry [10], most FHE schemes (such as BGV/BFV/CKKS [1] [7] [6]) are only specially designed for two basic homomorphic operations, namely addition and multiplication. This is mainly because all computation tasks can somehow be represented/approximated by a polynomial function, and thus can be evaluated using a series of homomorphic additions and multiplications. In terms of computational cost, homomorphic multiplication is typically much more complex and costly than homomorphic addition. Thus, for a given computation task, a natural goal is to find a functionally equivalent polynomial so that it can be homomorphically evaluated with fewer multiplication operations.

Currently, one of the most important computation tasks for almost all FHEs is to homomorphically evaluate their own decryption algorithm, which is also known as bootstrapping. The central part of bootstrapping is basically a polynomial to emulate the division or modular function, such as the digital extraction polynomials for the BGV/BFV-like schemes [4, 11] and the approximation polynomials for the CKKS-like schemes [3, 5, 14]. Since these polynomials usually have very large degrees, bootstrapping has become a major performance bottleneck for existing FHEs, for example, the time for bootstrapping a single BGV ciphertext is more than 26 minutes [13]. In the past decade, a large body of work [4, 9, 15, 16] have been devoted to improving the bootstrapping performance of FHEs in various parameter settings. A general way is to employ a SIMD-style bootstrapping by carefully exploiting the ring structure of the underlying ciphertexts so that a single ciphertext essentially contains a set of slots to encrypt different plaintexts. A more sophisticated way is to directly reduce the cost of bootstrapping by finding a functionally equivalent but can be efficiently evaluated polynomials.

Given a polynomial $f(x)$ with degree D, one can naturally apply the Horner's method to evaluate $f(x)$ at any input x with at most D multiplications. The number of multiplications can be further reduced to $O(\sqrt{D})$ by using the Paterson–Stockmeyer (P-S) method [17] at the cost of keeping $O(\sqrt{D})$ intermediate results during the computation. The above two methods can be applied to the evaluation of all polynomials. Recently, Okada, Player and Pohmann [16] showed that for some appropriate choices of parameters, there exist more efficient polynomial evaluation algorithms using the Galois structure of the plaintext slots. More specifically, let p, e, d be three positive integers, a plaintext slot in FHEs is typically defined by a Galois ring $\mathbb{Z}_{p^e}[x]/(F(x))$ where $F(x)$ is an irreducible polynomial modulo p of some degree d. The core technique of [16] is based on the assumption that any polynomial $f(X)$ with degree $D = d = 2^\ell$ defined over $\mathbb{Z}_{p^e}$ can be decomposed into a pair of polynomials $f(X) = h(X) + g(X)$ such that (1) $h(X)$ has degree exactly d but has a nice structure which can be evaluated with $O(\log d)$ multiplications; and (2) $g(X)$ has degree at most d and consists solely of monomials of power-of-two degrees, thus also admitting evalua-

tion with at most $O(\log d)$ multiplications. Okada, Player and Pohmann [16] also proposed an algorithm with $O(\log d)$ multiplications for polynomial $f(X)$ with degree $D < d = 2^\ell$, and showed that their algorithms can be used to speed up the bootstrapping of FHEs with appropriate choices of parameters. However, as commented in [18], selecting a suitable cyclotomic ring that has a desired degree d while also balancing security and efficiency is a highly constrained problem that often lacks an ideal solution. For example, neither $d = 2^\ell$ nor $D \leq d$ is satisfied for the BGV parameters in the recent work [15] (see Table 3), and the P-S method with $O(\sqrt{D})$ multiplications remains the state of the art for $d \neq 2^\ell$ or $D > d$.

1.1 Contributions

In this paper, we present asymptotically better algorithms for evaluating a degree-D polynomial $f(X)$ at a point $t \in \mathbb{Z}_{p^e}$, within the plaintext-slot structure $\mathbb{Z}_{p^e}[X]/(F(X))$. Our results hold for any arbitrary d and all $D < d \log p$, where p is prime and $F(X)$ is a monic irreducible polynomial of degree d. This is achieved by a careful generalization and combination of the ideas from Okada et al. [16] with the P-S method [17]. Our first algorithm is dedicated to the setting of $D \leq d$, which essentially works the same as that of Okada et al. [16] for $d = 2^\ell$, but also maintains the same asymptotic complexity, namely, $O(\log d)$ multiplications, for any non-power-of-two $d \neq 2^\ell$. Then, we extend the idea to the setting of $d < D < d \log p$, and present an algorithm with $O(\sqrt{D/d})$ multiplications which outperforms the P-S method [17] by a factor of $\sqrt{d}$. As the algorithms in [16], our algorithms also make use of certain heuristics for polynomial decomposition that, while only assumed to be reasonable, have been experimentally validated for all BGV parameters in [15].

Finally, we implement our algorithms to evaluate the digit extraction polynomials of the BGV bootstrapping in HElib, and give a concrete comparison with that of a recent work [15] in Table 1. We consider all the five parameter sets in [15], and give the major parameters related to our algorithms in the first five columns of Table 1, where p denotes the plaintext modulus, M denotes the index of the underlying cyclotomic rings, d and D are respectively the degree of the plaintext slots and the digit extraction polynomials. Our implementation is based on the publicly available source codes of [15][7], where the P-S method is used by default because of $d \neq 2^\ell$ and $d < D < d \log p$ for all the five parameter sets. We provide our source codes in the GitHub repository[8], and run the codes of ours and [15] on the same desktop computer equipped with an AMD Ryzen 9 9950X3D CPU and 64 GB of RAM. In the last three columns of Table 1, we present the average timing (over 100 times) for running the corresponding codes, which shows our algorithms are $1.22 - 2.16\times$ faster than that of [15].

[7] https://github.com/msh086/BGV-Boot-for-Large-p.
[8] https://github.com/Nobody673/artifact-helib/tree/main.

	Parameter Sets				Time (in seconds)		
ID	p	M	d	D	Ma et al. [15]	Ours	Speed-up
I	17	38309	24	81	274.48	225.04	1.22×
II	127	56647	45	135	1005.94	465.39	2.16×
III	257	55427	28	135	577.36	341.08	1.69×
IV	8191	45193	14	94	168.52	112.31	1.50×
V	65537	50731	18	94	270.59	149.96	1.80×

Table 1. Digit extraction performance compared to [15].

1.2 Technical Overview

We begin by first recalling the techniques to evaluate a polynomial $f(X)$ of degree $D \leq d = 2^\ell$ at some point $t \in \mathbb{Z}_{p^e}$, within the plaintext-slot structure $\mathbb{Z}_{p^e}[x]/(F(x))$ from Okada et al. [16], where p is prime and $F(X)$ is a monic irreducible polynomial of degree d. The key observation of [16] is that a degree-d monic irreducible polynomial $h(X)$ can be efficiently evaluated with $O(\log d)$ multiplications by using the Galois structure of $S = \mathbb{Z}_{p^e}[x]/(F(x))$.

In more detail, since $h(X)$ is a monic basic irreducible polynomial of degree d, we have that h has a root $\alpha \in S$, and its full set of roots in S is given by the Galois orbit $\{\alpha, \pi(\alpha), \pi^2(\alpha), \ldots, \pi^{d-1}(\alpha)\}$, where the iterates of π generate the full Galois group of the extension $\mathrm{Gal}(S/\mathbb{Z}_{p^e}) = \langle\pi\rangle$. Now fix $\alpha \in S, t \in \mathbb{Z}_{p^e}$ and consider $\alpha - t \in S$, define the shifted polynomial $h'(X) := h(X + t) \in \mathbb{Z}_{p^e}[X]$. Then $h'(\alpha - t) = h(\alpha) = 0$, and h' is monic of degree d. Moreover, h' is the minimal polynomial of $\alpha - t$ over $\mathbb{Z}_{p^e}$, and the roots of h' in S are the Frobenius conjugates of $\alpha - t$ so that $h'(X) = \prod_{i=0}^{d-1}\big(X - \pi^i(\alpha - t)\big)$. Applying Vieta's formula yields

$$h(t) = h'(0) \;=\; (-1)^d \prod_{i=0}^{d-1} \pi^i(\alpha - t).$$

Furthermore, for $d = 2^\ell$ and writing

$$A_0 := \alpha - t,\;\; A_{j+1} \;:=\; A_j \cdot \pi^{2^j}(A_j) \;\;\text{for } j = 0, \ldots, \ell - 1,$$

we have $A_\ell = \prod_{i=0}^{d-1} \pi^i(\alpha - t)$. This immediately gives us an algorithm which can evaluate $h(t)$ with exactly $\log d$ multiplications and $\log d$ automorphisms.

Evaluating Polynomials for $D \leq d \neq 2^\ell$. Our first observation is that $h(t)$ can still be efficiently evaluated with $O(\log d)$ multiplications when $d \neq 2^\ell$. Concretely, let $B_k = \prod_{i=0}^{k-1} \pi^i(\alpha - t)$ for $k \geq 1$ and $T(k)$ be the number of multiplications for computing B_k. We also define $T(0) = 0$ for ease of presentation. Then, for any $d = qd' + r$, by defining $\tilde{\pi} = \pi^q$ we have

$$B_d = \prod_{i=0}^{d'-1} \pi^{iq}(B_q) \cdot \pi^{qd'}(B_r) = \prod_{i=0}^{d'-1} \tilde{\pi}^i(B_q) \cdot \pi^{qd'}(B_r).$$

This means that by first computing B_q, B_r and then computing $\prod_{i=0}^{d'-1} \tilde{\pi}^i(B_q)$, we essentially can compute B_d by using at most

$$T(d) \leq T(d') + T(q) + T(r) + 1_{r \neq 0}$$

multiplications. Fixing $q = 2$, we have that $r \leq 1$. Using the facts $T(0) = T(1) = 0, T(2) = 1$, we have

$$T(d) \leq T(\lfloor d/2 \rfloor) + 1 + 1_{d \neq 2^\ell}.$$

This means that we have an algorithm evaluating $h(t) = B_d$ with at most $2 \log d$ multiplications. Moreover, the number of multiplications can be reduced to $T(d) = \log d$ for $d = 2^\ell$, which recovers the algorithm in [16]. One can also use $q = 3$ to optimize the total number of multiplications if d is a multiple of 3.

To extend the above algorithm to any polynomial $f(x)$ of degree d, we require the heuristic first introduced in [16]:

Heuristic 1 ([16]) *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ be a polynomial. Then there exists*

$$g = a_0 + \sum_i a_{i+1} X^{2^i} \in (\mathbb{Z}/p\mathbb{Z})[X]$$

of degree less than $\deg(f)$ *such that $f + g$ is a basic irreducible polynomial.*

The above heuristic assumption essentially says that any polynomial $f(X)$ of degree d can be decomposed into a basic irreducible polynomial $h(X)$ of degree d and a polynomial $g(X)$ which can be efficiently evaluated with at most $\log d$ multiplications. This immediately gives us an algorithm with $O(\log d)$ multiplications to evaluate any polynomial $f(X)$ of degree d. The techniques can also be extended to handle $f(X)$ with degree $D < d$ as that in [16].

Evaluating Polynomials for $d < D < d \log p$. Our second observation is that the Galois structure of $S = \mathbb{Z}_{p^e}[X]/(F(X))$ also enables efficient evaluation of the product of multiple monic basic irreducible polynomials. Concretely, let $H(X) = \prod_{j=0}^{k-1} h_j(X)$ be a polynomial of degree kd, where h_j are monic basic irreducible polynomials of degree d for $0 \leq j \leq k-1$. Let $\alpha_j \in S$ be the root of h_j. By the analysis given before, we have that for any $t \in \mathbb{Z}_{p^e}$, $h_j(t) = (-1)^d \prod_{i=0}^{d-1} \pi^i(\alpha_j - t)$. This implies that

$$H(t) = (-1)^{kd} \prod_{j=0}^{k-1} \prod_{i=0}^{d-1} \pi^i(\alpha_j - t).$$

Note that π is a ring homomorphism, by exchanging the outer and inner products we can obtain

$$H(t) = (-1)^{kd} \prod_{i=0}^{d-1} \pi^i \left(\prod_{j=0}^{k-1} (\alpha_j - t) \right).$$

Then, by applying the P-S method we can evaluate $C_k = \prod_{j=0}^{k-1}(\alpha_j - t)$ by using $O(\sqrt{k})$ multiplications. After obtaining C_k, by using the strategy given before we can evaluate $H(t) = (-1)^{kd}\prod_{i=0}^{d-1}\pi^i(C_k)$ with at most $O(\log d)$ multiplications. This immediately gives us an algorithm to evaluate $H(t)$ with at most $O(\sqrt{k} + \log d)$ multiplications.

To extend the above technique to any $d < D < d\log p$, we also need the following heuristic:

Heuristic 2 *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ be a polynomial of degree $D < d\log p$. Then, there exists a polynomial $g(X) = a_0 + \sum_i a_{i+1}X^{2^i} \in (\mathbb{Z}/p\mathbb{Z})[X]$ of degree less than $\deg(f)$ such that*

$$f(X) = a(X) \cdot \prod_{i=1}^{k} h_i(X) + g(X),$$

where each $h_i(X)$ is a degree-d monic irreducible polynomial and $a(X)$ is an arbitrary monic polynomial of degree $D - kd < d$ in $\mathbb{Z}_{p^e}[X]$.

Note that when $D = d$, the above heuristic essentially reduces to Heuristic 1 due to Okada et al. [16]. As discussed in [16], we are unable to provide a formal proof for Heuristic 2. However, based on a counting argument on the number of possible $g(X)$, we believe it is likely to hold when $D < d\log p$. This is the sole reason we impose the condition $D < d\log p$. We have also empirically verified Heuristic 2 via a brute-force search, and successfully found the required decompositions for the digit extraction polynomials used in BGV bootstrapping [15]. These experimental results give us strong confidence in its validity. We leave a formal proof of Heuristic 2 as an open problem for future work.

Given Heuristic 2 and the fact that $a(X)$ and $g(X)$ can be evaluated in $O(\log d)$ and $O(\log D)$ multiplications respectively, there exists an algorithm for evaluating a degree-D polynomial $f(X)$ for $d < D < d\log p$ using at most $O(\sqrt{D/d} + \log D) \leq O(\sqrt{\log p} + \log D)$ multiplications.

1.3 Organization

Section 2 introduces the notation and background used throughout, including the BGV scheme and prior methods for homomorphic polynomial evaluation. Section 3 develops our algorithms, including the orbit product evaluation schedule for general slot degrees, polynomial decomposition templates, and an optimization via the incorporation of null polynomials. Section 4 applies our algorithms to BGV bootstrapping, with a focus on accelerating the digit-extraction stage, and presents detailed implementation details and experimental evaluations of our approach in HElib.

2 Preliminaries

In this paper, we denote univariate monic polynomials by $f(X), g(X), h(X)$, $H(X)$, where X represents the input to the polynomials; A polynomial $f(X)$

over $\mathbb{Z}_{p^e}$ is said to be *basic irreducible* if and only if $\hat{f}(X) = f(X) \bmod p$ is irreducible over $\mathbb{F}_p$; Unless otherwise noted, the logarithm symbol log used in this paper is base-2 (i.e., $\log = \log_2$).

2.1 Algebraic structure of the plaintext space

We recall the notations used to describe the algebraic structure of the plaintext space in SIMD-style RLWE-based FHE schemes (using HElib's BGV instantiation as a concrete example [12]).

Symbol	Definition	Description
p	prime	Plaintext modulus base
e	positive integer	plaintext modulus exponent
M	positive integer	Cyclotomic index
m	value of $\varphi(M)$	Degree of the M-th cyclotomic polynomial
d	$\mathrm{ord}_{(\mathbb{Z}/M\mathbb{Z})^\times}(p)$	Degree of each irreducible factor
n	m/d	Number of irreducible factors of $\Phi_M(X)$ over $\mathbb{F}_p$
W	$\mathbb{Z}[x]/\Phi_M(x)$	Polynomial ring modulo $\Phi_M(x)$
S	$GR(p^e, d)$	Galois ring, degree-d extension of $\mathbb{Z}/p^e\mathbb{Z}$

Table 2. Notations for SIMD FHEs

Let M be the cyclotomic index and write $\Phi_M(X) \in \mathbb{Z}[X]$ for the M-th cyclotomic polynomial. We write x for the residue class of X modulo the defining polynomial and define the cyclotomic ring $W := \mathbb{Z}[x]/(\Phi_M(x))$. Fix a prime $p \nmid M$ and a prime-power plaintext modulus p^e. The plaintext ring used in HElib can be viewed as the reduction

$$W_{p^e} := W/(p^e) \cong (\mathbb{Z}/p^e\mathbb{Z})[x]/(\Phi_M(x)).$$

When $e = 1$, reducing modulo p yields $W_p \cong \mathbb{F}_p[x]/(\Phi_M(x))$. A standard fact is that $\Phi_M(X)$ factors over $\mathbb{F}_p$ as a product of distinct irreducibles

$$\Phi_M(X) = \bar{F}_1(X) \cdots \bar{F}_n(X) \bmod p \qquad \deg(\bar{F}_i) = d, \quad n = \varphi(M)/d,$$

where $d = \mathrm{ord}_{(\mathbb{Z}/M\mathbb{Z})^\times}(p)$ is the slot degree. By the polynomial CRT, $W_p \cong \prod_{i=1}^{n} \mathbb{F}_p[x]/(\bar{F}_i(x))$. Each factor is isomorphic to the finite field $\mathbb{F}_{p^d}$; we denote this common degree-d extension by E and call it the slot algebra. Under this identification, we have $W_p \cong E^n$, so a plaintext corresponds to an n-tuple of *slots* in E, and ring addition/multiplication in W_p implements component-wise (SIMD) operations on these slots.

For $e > 1$, each irreducible factor $\bar{F}_i(X) \in \mathbb{F}_p[X]$ admits a basic-irreducible lift $F_i(X) \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ of the same degree d. Consequently, we obtain a slot

decomposition

$$W_{p^e} \;=\; (\mathbb{Z}/p^e\mathbb{Z})[x]/(\Phi_M(x)) \;\cong\; \prod_{i=1}^{n} S_i, \qquad S_i \;:=\; (\mathbb{Z}/p^e\mathbb{Z})[x]/(F_i(x)),$$

where each S_i is a degree-d extension of $\mathbb{Z}_{p^e} := \mathbb{Z}/p^e\mathbb{Z}$, and in particular $S_i \cong GR(p^e, d)$ for all i. Thus, a plaintext in W_{p^e} can be viewed as an n-tuple of slots, with the i-th slot living in S_i.

For any $j \in (\mathbb{Z}/M\mathbb{Z})^\times$, the map $\theta_j : \; f(x) \longmapsto f(x^j)$ defines a ring automorphism of W_{p^e}. Under the above CRT identification, the family $\{\theta_j\}$ induces structured permutations of the slots. In particular, let $\bar{p} = p \bmod M$. The special automorphism $\theta_{\bar{p}}$ restricts on each slot S_i to the generalized Frobenius automorphism π_i. Fix one slot, say S_1, and write $\pi := \pi_1$. Then the iterates of π generate the full Galois group of the extension:

$$\mathrm{Gal}(S_1/\mathbb{Z}_{p^e}) \;=\; \langle \pi \rangle \;\cong\; C_d.$$

Let $\alpha \in S_1$ be the residue class of x (equivalently, a root of F_1 in S_1). The elements $\pi^i(\alpha)$ for $i = 0, \ldots, d-1$ form the Galois orbit of α, i.e., the d Frobenius conjugates of this slot generator. This orbit viewpoint is exactly what underlies the Trace/Norm operators used later: Trace is the orbit sum and Norm is the orbit product over $\mathrm{Gal}(S_1/\mathbb{Z}_{p^e})$. In the rest of the paper, for the sake of simplicity, we restrict our analysis to the behavior of a single plaintext slot; when referring to S, we denote a non-specific slot algebraic structure (i.e., $GR(p^e, d)$).

2.2 Norm and Trace

As discussed above, in the SIMD-style RLWE-based setting (e.g., BGV/BFV), plaintexts live in an extension of the base ring/field (e.g., $\mathbb{F}_p$ or $\mathbb{Z}_{p^e}$), and the packed-slot representation naturally exposes a Galois structure over base ring/field via available automorphisms. Okada *et al.* [16] exploited this structure and proposed Norm/Trace-based algorithms for fast evaluation of structured polynomials, achieving $O(\log d)$ multiplicative complexity in the single-component setting, where d denotes the slot degree. For $\alpha \in S$, define the trace and norm from S down to $\mathbb{Z}_{p^e}$ by

$$\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\alpha) \;:=\; \sum_{\sigma \in \mathrm{Gal}(S/\mathbb{Z}_{p^e})} \sigma(\alpha), \qquad \mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha) \;:=\; \prod_{\sigma \in \mathrm{Gal}(S/\mathbb{Z}_{p^e})} \sigma(\alpha).$$

Since $\mathrm{Gal}(S/\mathbb{Z}_{p^e}) = \langle \pi \rangle$ has order d, the equation above can be rewritten as

$$\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\alpha) \;=\; \sum_{i=0}^{d-1} \pi^i(\alpha), \qquad \mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha) \;=\; \prod_{i=0}^{d-1} \pi^i(\alpha).$$

Let $h(X) \in \mathbb{Z}_{p^e}[X]$ be a monic *basic irreducible* polynomial of degree d (i.e., its reduction modulo p is irreducible over $\mathbb{F}_p$). Then h has a root $\alpha \in S$, and

h is the minimal polynomial of α, so its full set of roots in S is given by the Galois orbit $\{\alpha, \pi(\alpha), \pi^2(\alpha), \ldots, \pi^{d-1}(\alpha)\}$. Since h is the minimal polynomial of α over $\mathbb{Z}_{p^e}$ and splits in S, we have $h(X) = \prod_{i=0}^{d-1}\big(X - \pi^i(\alpha)\big)$. By Vieta's formula, the constant term satisfies

$$h(0) \;=\; (-1)^d \prod_{i=0}^{d-1} \pi^i(\alpha) \;=\; (-1)^d \,\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha).$$

Now fix $\alpha \in S,\ t \in \mathbb{Z}_{p^e}$ and consider $\alpha - t \in S$, define the shifted polynomial $h'(X) \;:=\; h(X+t) \in \mathbb{Z}_{p^e}[X]$. Then $h'(\alpha - t) = h(\alpha) = 0$, and h' is monic of degree d. Moreover, h' is the minimal polynomial of $\alpha - t$ over $\mathbb{Z}_{p^e}$. Consequently, the roots of h' in S are the Frobenius conjugates of $\alpha - t$ so that: $h'(X) = \prod_{i=0}^{d-1}\big(X - \pi^i(\alpha - t)\big)$. Applying Vieta's formula again yields

$$h'(0) \;=\; (-1)^d \prod_{i=0}^{d-1} \pi^i(\alpha - t) \;=\; (-1)^d \,\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha - t).$$

Since $h'(X) \;=\; h(X+t)$, we can obtain:

$$h(t) \;=\; h'(0) \;=\; (-1)^d \,\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha - t).$$

Finally, the norm can be evaluated as a full Galois orbit product

$$\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha - t) \;=\; \prod_{i=0}^{d-1} \pi^i(\alpha - t).$$

Furthermore, by assuming $d = 2^\ell$ and writing

$$A_0 := \alpha - t,\ A_{j+1} \;:=\; A_j \cdot \pi^{2^j}(A_j) \;\text{ for } j = 0, \ldots, \ell - 1,$$

we have $A_\ell = \prod_{i=0}^{d-1} \pi^i(\alpha - t) = \mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(\alpha - t)$. This immediately gives us an algorithm which can evaluate $h(t)$ with exactly $\log d$ ciphertext–ciphertext multiplications and $\log d$ automorphisms, and has multiplicative depth $\log d$.

2.3 Polynomial Evaluations Using Norm and Trace

Given the above facts, Okada, Player and Pohmann [16] provided efficient algorithms to evaluate $f(X) \in \mathbb{Z}_{p^e}[X]$ of degree $D \le d$ with $O(\log d)$ multiplications based on several heuristics on polynomial decomposition.

Heuristic 3 ([16]) *Let $f \in (\mathbb{Z}/p^e\mathbb{Z})[X]$ be a polynomial. Then there exists*

$$g = a_0 + \sum_i a_{i+1} X^{2^i} \in (\mathbb{Z}/p\mathbb{Z})[X]$$

of degree less than $\deg(f)$ *such that $f + g$ is basic irreducible.*

Using the above heuristic, for any polynomial $f(X) \in \mathbb{Z}_{p^e}[X]$ of degree $D = d$, there exists a basic irreducible polynomial $h(X)$ of degree d and a polynomial $g = a_0 + \sum_{i+1} a_i X^{2^i}$ such that $f = h - g$. Thus, $f(t)$ can be evaluated for any $t \in \mathbb{Z}_{p^e}$ by evaluating $h(t)$ and $g(t)$ separately. Given the facts in the above subsection, $h(t)$ can be evaluated with $O(\log d)$ multiplications. For the $g(t) = a_0 + \sum_{i+1} a_i X^{2^i}$ component, it suffices to compute the sequence $t^2,\ t^4,\ t^8,\ \ldots,\ t^{2^{l-1}}$ by repeated squaring, and then combine them using additions and plaintext multiplications. This requires at most $\log d - 1$ ciphertext multiplications.

Lemma 1 (Norm Method [16]). *Let $f(X) \in \mathbb{Z}_{p^e}[X]$ be an arbitrary monic polynomial of degree $D = d = 2^\ell$. Then, under heuristic 3, there exists an algorithm evaluating $f(t)$ for any $t \in \mathbb{Z}_{p^e}$ with at most $2\log d$ ciphertext multiplications and $\log d$ automorphisms. Moreover, the multiplicative depth is $\log d$.*

For the arbitrary monic polynomial f of degree $D < d = 2^\ell$, under the heuristic below:

Heuristic 4 ([16]) *There exists a monic basic-irreducible polynomial of the special form*

$$h(X) = X^d + b \in (\mathbb{Z}/p^e\mathbb{Z})[X],\ b \in \mathbb{Z}_{p^e}, \tag{2.3.1}$$

and let $\alpha \in S$ be a root of h (so h is the minimal polynomial of α over S).Then the power-sum traces satisfy

$$\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\alpha^t) \ = \ \begin{cases} d, & t = 0, \\ 0, & 1 \le t \le d-1. \end{cases}$$

Consequently, for any polynomial $f(X) = \sum_{j=0}^{D} a_j X^j \in \mathbb{Z}_{p^e}[X],\ a_j \in \mathbb{Z}_{p^e}$ of degree $D < d = 2^\ell$ and any $t \in \mathbb{Z}_{p^e}$, one can recover $f(t)$ from a single trace computation as follows. Let

$$L \ := \ \lceil \log D \rceil,\ \beta \ := \ \prod_{i=0}^{L-1} \left(1 + (\alpha t)^{2^i}\right),\ \gamma \ := \ \left(\sum_{j=0}^{D} a_j\, \alpha^{-j}\right) \cdot \beta.$$

Then

$$\frac{1}{d}\ \mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma) \ = \ \sum_{j=0}^{2^L-1} a_j t^j \ = \ f(t), \qquad \text{where } a_j = 0 \text{ for } j > D. \tag{2.3.2}$$

Computing β uses $2L - 2 = 2\lceil \log D \rceil - 2$ ciphertext multiplications (by repeated squaring of αt and multiplying the factors), and computing $\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma) = \sum_{i=0}^{d-1} \pi^i(\gamma)$ uses $\log d$ automorphisms.The overall cost is described in the lemma below:

Lemma 2 (Trace Method [16]). *For an arbitrary monic polynomial $f(X) \in \mathbb{Z}_{p^e}[X]$ of degree $D < d = 2^\ell$, and $t \in \mathbb{Z}_{p^e}$.One can evaluate $f(t)$ in $\lceil 2\log D \rceil - 2$ ciphertext multiplications and $\log d$ automorphisms under heuristic 4 .The multiplicative depth is $\lceil \log D \rceil$.*

2.4 The Paterson–Stockmeyer Method

The most general-purpose approach is the Paterson-Stockmeyer (P-S) algorithm [17], which applies over arbitrary rings/fields and evaluates a degree-D polynomial using $2\sqrt{D} + O(1)$ ciphertext–ciphertext multiplications and multiplicative depth $\sqrt{D}+O(\log D)$. Later refinements also provide an optimal-depth variant that achieves $O(\log D)$ multiplicative depth with multiplication count $2\sqrt{D} + O(\log D)$ [17].

Lemma 3 (Paterson-Stockmeyer [17]). *Let f be a polynomial of degree D $f(X) = \sum_{i=0}^{D} a_i X^i$, we aim to evaluate at $t \in R$. Let $m = \lfloor \sqrt{D} \rfloor$, write*

$$f(t) = \sum_{j=0}^{\lceil D/m \rceil - 1} t^{jm} f_j(t), \qquad f_j(t) = \sum_{s=0}^{m-1} a_{jm+s} t^s.$$

Then $f(t)$ can be evaluated using $(m-1)+(\lceil D/m \rceil - 1) = 2\sqrt{D}+O(1)$ non-scalar multiplications*, and multiplicative depth is $(m-1) + \lceil \log (\lceil D/m \rceil) \rceil = O(\sqrt{D})$.*

Lemma 4 (Optimal-Depth Paterson-Stockmeyer [17]). *One may precompute baby-step powers(stride powers)*

$$t^m,\ t^{2m},\ t^{4m},\ \ldots,\ t^{2^l m} \quad \textit{where } l = \lfloor \log (\lceil D/m \rceil - 1) \rfloor,$$

and evaluate the outer polynomial in t^m via a balanced recursion, yielding multiplicative depth of $\log D\ + O(1)$ and total non-scalar multiplications $2\sqrt{D} + \frac{1}{2}\log D + O(1)$.

3 Faster Algorithms for Polynomial Evaluations

In the previous section, we reviewed the Norm/Trace-based polynomial evaluation methods of Okada [16], which exploit the algebraic structure of the plaintext slot algebra. In HElib, however, these methods are difficult to instantiate for two main reasons: (1) they assume a power-of-two slot degree $d = 2^\ell$, which is difficult to guarantee in HElib; and (2) they apply only to target polynomials of degree D satisfying $D \leq d$. In this section, we present two new algorithms by carefully generalizing and combining the ideas of Okada et al. with the Paterson–Stockmeyer (P-S) method. Our first algorithm is tailored to the setting of $D \leq d$: it retains the same multiplicative complexity as Okada et al.'s algorithm while lifting the power-of-two restriction on d. Using the same line of reasoning, we construct a second algorithm for evaluating polynomials with degrees in the range $d < D \leq d \log p$, which requires $O(\sqrt{D/d})$ multiplications and outperforms the P-S method by a factor of $\sqrt{d}$.

In Section 3.1, we first extend the Norm method from the idealized setting of $d = 2^\ell$ to general d, thereby overcoming the first obstacle. In Section 3.2, we progressively expand the degree range handled by Norm-based evaluation: from the base case of $D = d$ to $D \leq d \log p$. We then examine the plausibility of the existence hypothesis in Section 3.3. In Section 3.4, we incorporate null polynomials as an optimization to further reduce the multiplication cost.

3.1 Evaluating Polynomials of degree $D \leq d$ for arbitrary d

Let $\mathrm{Gal}(S/\mathbb{Z}_{p^e}) = \langle \pi \rangle$ be a cyclic Galois group of size d, and let $P(t) \in S$ denote the plaintext expression, where $t \in \mathbb{Z}_{p^e}$ is the input to the target polynomial, whose full-orbit product (with generator π) we aim to compute: $\mathrm{Orb}_d^{(\pi)}(P(t)) := \prod_{j=0}^{d-1} \pi^j(P(t))$, which is another notation for $\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(P(t))$.

To handle general (non power-of-two) d, we use a recursive divide-with-remainder regrouping. This regrouping is closely related to classical Frobenius-map scheduling for trace computations, in particular the work of von zur Gathen and Shoup [8]. In this work, we adapt this scheduling to homomorphic norm evaluation. Fix an integer $q \geq 2$ and write $d = qd' + r,\ 0 \leq r < q$, and define the block and remainder products

$$B := \mathrm{Orb}_q^{(\pi)}(P(t)) = \prod_{u=0}^{q-1} \pi^u(P(t)), \quad C := \mathrm{Orb}_r^{(\pi)}(P(t)) = \prod_{u=0}^{r-1} \pi^u(P(t)),$$

where $C := 1$ if $r = 0$. Using the multiplicativity of Galois automorphisms and partitioning the orbit into d' consecutive blocks of length q, we derive

$$\mathrm{Orb}_d^{(\pi)}(P(t)) = \left(\prod_{k=0}^{d'-1} \pi^{kq}(B) \right) \cdot \pi^{qd'}(C) = \mathrm{Orb}_{d'}^{(\pi^q)}(B) \cdot \pi^{qd'}(C). \tag{3.1.1}$$

Therefore, computing a size-d orbit product reduces to evaluating three smaller orbit products: (i) $\mathrm{Orb}_{d'}^{(\pi^q)}(B)$ (size d' under generator π^q); (ii) $B = \mathrm{Orb}_q^{(\pi)}(P(t))$ (size q); (iii) $C = \mathrm{Orb}_r^{(\pi)}(P(t))$ (size r) followed by one final multiplication if $r \neq 0$. We then evaluate each of these orbit products recursively using the same divide-with-remainder rule.

Let $T(d)$ denote the number of homomorphic ciphertext multiplications required by our schedule to evaluate $\mathrm{Orb}_d^{(\cdot)}(P)$, and this count is independent of the choice of generator. From (3.1.1) we obtain the recursion $T(d) \leq\ T(d') + T(q) + T(r) + \mathbf{1}_{\{r \neq 0\}},\ d = qd' + r,\ 0 \leq r < q$, where the indicator accounts for the final multiplication by $\pi^{qd'}(C)$ when $r \neq 0$. Clearly, the initial values for this recurrence relation are: $T(0) = T(1) = 0,\ T(2) = 1,\ T(3) = 2$. We adopt the following rule to select q:

- If $2 \mid d$, set $q = 2$, yielding $d' = d/2$ and $r = 0$.
- If $3 \mid d$ (and $2 \nmid d$), set $q = 3$, yielding $d' = d/3$ and $r = 0$.
- Otherwise set $q = 2$, so $d' = \lfloor d/2 \rfloor$ and $r = 1$.

Under this rule, the largest recursive subproblem has size at most half that of the original problem at each level, and each level incurs at most a constant number of homomorphic ciphertext multiplications—i.e., up to 2 from forming B, and up to 1 from final multiplication if $r \neq 0$—with these two costs not occurring simultaneously. This yields the following bound on the multiplicative complexity of the orbit product:

$$T(d)\ \leq\ T\left(\frac{d}{2}\right) + 2\ \leq\ 2\lceil \log d \rceil, \tag{3.1.2}$$

which generalizes to logarithmic multiplicative complexity for arbitrary d.

From the perspective of multiplicative depth, each recursion increases the multiplicative depth by 1 to 2 levels (increasing by 1 level if $2 \mid d$, and by 2 levels otherwise); in other words, the original component has a multiplicative depth that is 1 to 2 levels higher than its recursively derived subcomponents. This gives a corresponding upper bound on the multiplicative depth:

$$\mathrm{Depth}(d) \leq \mathrm{Depth}\left(\frac{d}{2}\right) + 2 \leq 2\lceil \log d \rceil \tag{3.1.3}$$

Proposition 1 (Cost Bounds for OrbitProduct). *Let* $\mathrm{Orb}_d^{(\pi)}(P(t))$ *denote the orbit product for* $P(t) \in S$ *with arbitrary degree* d*. Building on the recursive bounds in Equations* (3.1.2) *and* (3.1.3)*, evaluating* $\mathrm{Orb}_d^{(\pi)}(P(t))$ *requires at most*

$$2\lceil \log d \rceil$$

homomorphic ciphertext–ciphertext multiplications, with a multiplicative depth of at most:

$$2\lceil \log d \rceil$$

When feasible, choosing 2- and 3-smooth values of d further reduces the constant factors in (3.1.2); additionally, 2- and 3-smooth values are also favorable for multiplicative depth (3.1.3). For values of d of approximately the same magnitude, the case where $d = 2^\ell$ achieves the optimal multiplicative depth and multiplicative complexity. In the BGV scheme, however, d is determined by the cyclotomic parameter M and constrained by security requirements, noise growth, and SIMD packing considerations; thus, requiring d to be a pure power of two is often impractical.

Our divide-with-remainder strategy can be applied not only to computing $\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(P(t))$ for $P(t) \in S$, but also to computing $\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma)$ for $\gamma \in S$ structured as in Lemma 2—this is because Galois automorphisms are also additive homomorphisms. Specifically, we compute the orbit addition in $\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma)$ following the same rules used to compute the orbit product in $\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(P(t))$, with only the multiplication operators replaced by addition operators.

However, when $d \neq 2^\ell$, γ in $\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma)$ requires distinct treatment: in the computation of $\gamma = \left(\sum_{j=0}^{D} a_j\, \alpha^{-j}\right) \cdot \beta$, we use $\beta = \sum_{i=0}^{D} X^i$ instead of $\beta = \prod_{i=0}^{L-1}\left(1 + (\alpha t)^{2^i}\right)$ (as defined in Lemma 2). Fortunately, $\beta = \sum_{i=0}^{D} X^i$ can still be computed with a multiplicative complexity of at most $\lfloor 3 \log D \rfloor$ with the depth of $\lfloor 2 \log D \rfloor$; as this result is tangential to the main line of reasoning, we defer its proof to Appendix A Proposition 4.

We have described how to compute $\mathrm{Norm}_{S/\mathbb{Z}_{p^e}}(P(t))$ (and $\mathrm{Tr}_{S/\mathbb{Z}_{p^e}}(\gamma)$) for non-powers-of-two values of d and $P(t) \in S (\gamma \in S)$. Building on Heuristic 3 (and Heuristic 4), we can therefore efficiently evaluate any monic polynomial $f(X)$ of degree $D \leq d$ when d is a non-power-of-two—by decomposing it as $f(X) = h(X) + g(X)$ when $D = d$(or constructing β and γ as in Lemma 2 when $D < d$). We thus arrive at the following proposition:

Proposition 2 (Cost Bounds for Polynomial Evaluation with $D \leq d$ (Arbitrary d)). *Let $f(X) \in \mathbb{Z}_{p^e}[X]$ be a target polynomial of degree $D = d$. Evaluating $f(X)$ at $t \in \mathbb{Z}_{p^e}$ requires at most*

$$\underbrace{2\lceil \log d \rceil}_{Orbit\ Product} + \underbrace{\lfloor \log d \rfloor}_{Handling\ g(t)}$$

homomorphic ciphertext–ciphertext multiplications, with a multiplicative depth of at most:

$$2\lceil \log d \rceil.$$

For a target polynomial $f(X) \in \mathbb{Z}_{p^e}[X]$ of degree $D < d$, evaluating $f(X)$ at $t \in \mathbb{Z}_{p^e}$ requires at most

$$3\lfloor \log D \rfloor$$

homomorphic ciphertext–ciphertext multiplications, with a multiplicative depth of at most:

$$\lfloor 2 \log D \rfloor.$$

3.2 Evaluating degree-D Polynomials for $d < D \leq d \log p$

We now consider a target polynomial $f(X) \in \mathbb{Z}_{p^e}[X]$ of degree $D \leq d \log p$ (we provide a detailed analysis of the origin of this upper bound in Section 3.3). We relax the requirement that the structured component must be exactly a degree-d basic irreducible polynomials; instead, we only require that it contains $k = \lfloor D/d \rfloor$ distinct degree-d basic irreducible factors. Concretely, we seek decompositions of the form

$$f(X) = a(X) \cdot \prod_{i=1}^{k} h_i(X) + g(X), \tag{3.2.1}$$

where the auxiliary polynomial $a(X) \in \mathbb{Z}_{p^e}[X]$ absorbs the residual part of $f(X)$.Our method for evaluating this target polynomial relies on a heuristic:

Heuristic 5 *There exists a polynomial*

$$g(X) = a_0 + \sum_i a_{i+1} X^{2^i} \in (\mathbb{Z}/p\mathbb{Z})[X]$$

of degree less than $\deg(f)$ *such that* (3.2.1) *holds, where each $h_i(X)$ is a degree-d basic irreducible polynomial and $a(X)$ is an arbitrary monic polynomial of degree $D - kd$ in $\mathbb{Z}_{p^e}[X]$.*

We first demonstrate how to efficiently compute the component $H(X) = \prod_{i=1}^{k} h_i(X)$. For each i, let α_i denote a root of $h_i(X)$, so that evaluating $h_i(X)$ at $t \in \mathbb{Z}_{p^e}$ can be expressed as a Galois orbit product (with a factor of $(-1)^d$):

$$h_i(t) = (-1)^d \operatorname{Orb}_d^{(\pi)}(\alpha_i - t).$$

A naive approach would compute each orbit product $\mathrm{Orb}_d^{(\pi)}(\alpha_i - t)$ separately and then multiply the results. Instead, we swap the order of multiplication by leveraging the multiplicativity of Galois automorphisms. We first form the base product $P(t) := \prod_{i=1}^{k}(\alpha_i - t)$, which we evaluate using the optimal-depth P-S algorithm (Lemma 4) **with** $2\sqrt{k} + \frac{1}{2}\log k + O(1)$ **multiplications and a depth of** $\log k + O(1)$**.** We then apply the method introduced in Section 3.1 to evaluate the orbit product once:

$$H(t) = \prod_{i=1}^{k} \mathrm{Orb}_d^{(\pi)}(\alpha_i - t) = \mathrm{Orb}_d^{(\pi)}(P(t)) = \prod_{j=0}^{d-1} \pi^j\Big(P(t)\Big).$$

with **at most** $2\lceil \log d \rceil$ **multiplications** and a **depth of** $2\lceil \log d \rceil$**.**

We evaluate $g(t)$ via repeated squaring and addition as before with **at most** $\lfloor \log D \rfloor \le \log d + \log k$ **multiplications.** The remaining step is to compute $a(t)$ and combine the terms as $f(t) = a(t) \cdot H(t) + g(t)$. In principle, $a(t)$ can be evaluated using the Trace method (Lemma 2), which provides a convenient upper bound for our asymptotic analysis. In practice, however, we observe that $\deg(a)$ is typically small. Furthermore, the evaluation of the sparse polynomial $g(t)$ and base product $P(t)$ already generates a set of low-degree powers that can be reused when evaluating $a(t)$ via the Paterson–Stockmeyer (P-S) algorithm (Lemma 3); we therefore evaluate $a(t)$ using the P-S algorithm to leverage this reuse and reduce wall-clock latency. In our asymptotic analysis, we nevertheless upper-bound the cost of evaluating $a(t)$ by the complexity of the Trace method (Lemma 2). As a result, the remainder-handling **multiplication count is at most** $3\lfloor \log(D - kd) \rfloor < 3 \log d$**.** On the other hand, since the degree of $a(X)$ satisfies $D - kd < d$, its multiplicative depth does not exceed that of $H(X)$. Combining it with $H(X)$ thus **increases the multiplicative depth by only 1** relative to that of $H(X)$.

Proposition 3 (Cost Bound for Polynomial Evaluation with $d < D \le d \log p$**).** *Let* $f(X) \in \mathbb{Z}_{p^e}[X]$ *be a target polynomial of degree* $D \le d \log p$*. Evaluating* $f(X)$ *at* $t \in \mathbb{Z}_{p^e}$ *requires at most*

$$\underbrace{2\sqrt{k} + \frac{1}{2}\log k}_{Base\ Product} + \underbrace{3\lfloor \log d \rfloor}_{Handling\ a(t)} + \underbrace{2\lceil \log d \rceil}_{Orbit\ Product} + \underbrace{\log d + \log k}_{Handling\ g(t)}$$

homomorphic ciphertext–ciphertext multiplications, which is no more than

$$2\sqrt{D/d} + \frac{3}{2}\log D + \frac{9}{2}\log d + O(1) = 2\sqrt{D/d} + O(logD).$$

Calculate the Root $\boldsymbol{\alpha_i}$

In order to instantiate the base product we require plaintext constants $\alpha_1, \ldots, \alpha_k \in S$, which depend only on the plaintext parameters and the fixed target polynomial $f(X)$. Rather than working directly over $\mathbb{Z}_{p^e}$, we perform the algebraic

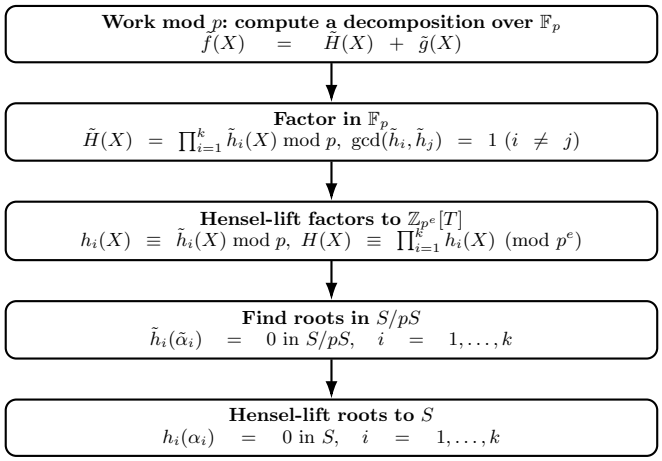


Fig. 1. Workflow for computing the plaintext constants $\{\alpha_i\}$.

steps modulo p and then lift to precision p^e using Hensel lifting. The rationale for this approach is algorithmic: irreducibility testing, factorization, and root-finding are substantially more standard and efficient over $\mathbb{F}_p$, whereas performing the same tasks natively over $\mathbb{Z}_{p^e}$ is considerably less convenient.

Concretely, we first calculate a decomposition over $\mathbb{F}_p$ to obtain $f \bmod p = \tilde{f} = \tilde{H} + \tilde{g}$, where $\tilde{g}$ is supported on power-of-two monomials and $\tilde{H}$ has the desired factor structure. We then factorize $\tilde{H}$ over $\mathbb{F}_p[X]$ as

$$\tilde{H}(X) = \prod_{i=1}^{k} \tilde{h}_i(X), \qquad \gcd(\tilde{h}_i, \tilde{h}_j) = 1 \quad (i \neq j).$$

The pairwise-coprimality requirement is essential for the standard Hensel factor-lifting step used here. We then find a root of each $\tilde{h}_i(X)$ in S/pS, lift the resulting factors to $\mathbb{Z}_{p^e}[X]$, and finally lift appropriate roots from S/pS to S to obtain the constants $\{\alpha_i\}$. This workflow is summarized in Figure 1.

3.3 On the Validity of the Hypothesis

Equations (3.2.1) posit a structured decomposition of the target polynomial into two components: (i) a product of k distinct degree-d basic irreducible factors (up to a low-degree auxiliary multiplier $a(X)$), and (ii) a power-of-two sparse correction polynomial $g(X)$. Rather than directly proving Hypothesis 5 in full generality, we formalize the following assumption—one supported by extensive empirical evidence from our polynomial decompositions over the parameter sets adopted in [4, 13, 15].

Definition 1 (Decomposition Type). *The decomposition type of a monic polynomial $f(X) \in \mathbb{F}_p[X]$ is the degree distribution of its irreducible factors,*

denoted as $\mathsf{decom}(f) = \{(d_1, k_1), \ldots, (d_m, k_m)\}$. *Here,* d_j *denotes the degree of an irreducible factor, and* k_j *denotes the number of distinct irreducible factors of* $f(X)$ *with degree* d_j.

We define the set of target decomposition types as

$$\mathsf{TargetDecomp} = \{\{(d, k), \dots\}, \{(d, k), \dots\}, \dots\},$$

where the only requirement is that the decomposition type contains exactly $k = \lfloor deg(f)/d \rfloor$ distinct irreducible factors of degree d (the characteristics of remaining factors are irrelevant to our method). We denote the set of all monic polynomials satisfying these decomposition types as

$$\mathsf{Satisfied} = \{H^\star(X) \in \mathbb{F}_p[X] \mid \deg(H^\star) = D \text{ and } \mathsf{decom}(H^\star) \in \mathsf{TargetDecomp}\}.$$

Our goal is to find a correction polynomial $g(X) \in \mathbb{F}_p[X]$ such that $f(X)(mod\, p) + g(X) = H^\star(X) \in \mathsf{Satisfied}$. Let the target polynomial $f(X)$ have degree D, and define the full space as the set of all monic degree-D polynomials over $\mathbb{F}_p$.

Lemma 5 (Gauss's Formula [2]). *The number of monic irreducible polynomials of degree* n *over* $\mathbb{F}_p$ *is*

$$N(p, n) = \frac{1}{n} \sum_{k|n} \mu(k) \cdot p^{n/k},$$

where $\mu(k)$ *denotes the Möbius function. For large* n, *this simplifies to the approximation* $N(p, n) \approx \frac{p^n}{n}$.

Theorem 6 (Size of the Satisfied Set). *The size of* Satisfied *(within the full space of monic degree-D polynomials over* $\mathbb{F}_p$*) is approximately* $\frac{p^D}{d^k k!}$.

Proof. By Lemma 5, the number of monic irreducible polynomials of degree d over $\mathbb{F}_p$ is approximately $\frac{p^d}{d}$. The number of distinct products of k distinct monic irreducible polynomials of degree d over $\mathbb{F}_p$ is

$$\frac{1}{k!} \cdot \frac{p^d}{d} \cdot (\frac{p^d}{d} - 1).. \cdot (\frac{p^d}{d} - k + 1) \approx \frac{p^{kd}}{d^k k!},$$

where the factor $\frac{1}{k!}$ removes orderings of the k factors that yield identical products. Any polynomial in Satisfied can be constructed by multiplying such a kd-degree product by an arbitrary monic polynomial of degree $D - kd$ over $\mathbb{F}_p$. There are exactly p^{D-kd} such monic degree-$D-kd$ polynomials, and a bijection exists between Satisfied and the pairs of (kd-degree product, $D-kd$-degree monic polynomial). Thus, the size of Satisfied is

$$\frac{p^{kd}}{d^k k!} \cdot p^{D-kd} = \frac{p^D}{d^k k!},$$

which completes the proof.

The density of the desired decomposition types within the full space is therefore

$$\rho = \frac{1}{d^k k!}. \tag{3.3.1}$$

We make the following key assumption: this density is preserved on the affine linear subspace of the form $f(X) + G$, where G denotes the linear subspace of sparse power-of-two polynomials.

Definition 2 (Family of Sparse Power-of-Two Polynomials G). *We define G as the family of sparse power-of-two polynomials, i.e.,*

$$G = \left\{ g(X) = a_0 + \sum_{i=0}^{\lfloor \log D \rfloor} a_{i+1} X^{2^i} \mid a_j \in \mathbb{F}_p,\ j = 0, 1, \ldots, \lfloor \log D \rfloor \right\}.$$

Under this assumption, a uniformly random $g \in G$ succeeds with probability $\approx 1/(d^k k!)$. Hence after about $d^k k!$ trials we expect to find a valid g. In particular, if $|G| \gg d^k k!$, we expect solutions to exist and to be easy to find by sampling.

The Moderate Range $D \leq d \log p$ The number of available monomials in the sparse correction polynomial $g(X)$ grows only logarithmically with the target degree $D = \deg(f)$, whereas the ratio D/d grows linearly with D for a fixed slot degree d. Consequently, when D/d becomes sufficiently large, the sparse family may not provide enough free terms to ensure that

$$f(X)(mod\ p) + g(X) \in \mathsf{Satisfied}$$

under the given density criterion (Eq. (3.3.1))—i.e., we must satisfy the following criterion:

$$p^{\lfloor \log D \rfloor + 2} \geq d^k \cdot k!, \qquad k = \left\lfloor \frac{D}{d} \right\rfloor. \tag{3.3.2}$$

Using the approximation $\log(k!) \leq k \log k$ and $dk \leq D$, we derive the following simplified criterion:

$$D \leq d \log p \cdot \left(1 + \frac{1}{\log D}\right),$$

which can be further approximated as:

$$D \leq d \log p. \tag{3.3.3}$$

Remark 1 (Empirical Validation). In our HElib implementation, we decompose the magic polynomial of highest degree (used during bootstrapping) as $f(X) = a(X)H(X) + g(X)$ for all parameter sets (p, e, d) with $p > 2$ considered in [4, 13, 15]. We successfully found such a decomposition for every parameter set, providing strong empirical evidence that our heuristics hold in practice-relevant regimes. Since the decomposition results are voluminous, we have uploaded the corresponding files to GitHub alongside our source code. A general algebraic proof of the existence of such decompositions is left to future work.

Search Strategy for $g(X)$ In our implementation, $g(X)$ is found via a simple compile-time enumeration over the family $\mathcal{G}$: we iterate over candidates in increasing order of the largest exponent, stopping at the first $g(X)$ that yields a valid decomposition of the form (3.2.1). We did not identify a consistently superior general-purpose strategy for finding $g(X)$ across different parameter sets and target polynomials. Designing more efficient $g(X)$-search procedures (or proving conditions under which the search terminates quickly) is an interesting direction for future research.

To summarize, we have extended our polynomial evaluation method to monic polynomials of arbitrary degree in a moderate range. In the next section, we introduce an optimization based on null polynomials to further reduce the online computational runtime.

3.4 An Optimization via Null Polynomials

We present an optimization for handling target polynomials of degree $D \leq d \log p$ by exploiting the existence of null polynomials modulo $\mathbb{Z}_{p^e}$—that is, nonzero polynomials that vanish for all inputs in the plaintext domain. Adding such a polynomial to a target polynomial $f(X)$ leaves its pointwise behavior over $\mathbb{Z}_{p^e}$ unchanged, while enabling us to adjust the polynomial's degree to a value more conducive to our decomposition strategy.

Null Polynomials over the Full Domain $\mathbb{Z}_{p^e}$ [9] Let $v_p(\cdot)$ denote the p-adic valuation, defined as

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

Let $\mathcal{O}$ denote the set of nontrivial null polynomials modulo p^e, specifically, polynomials $O(X) \in \mathbb{Z}_{p^e}[X]$ satisfying:

$$O(t) \equiv 0 \pmod{p^e} \text{ for all } t \in \mathbb{Z}_{p^e}.$$

We write $(X)_i := X(X-1)\cdots(X-i+1)$ for the i-th falling factorial and define the canonical family of null polynomials as $O_i(X) := p^{\max\{e - v_p(i!),0\}}\,(X)_i$. Every null polynomial modulo p^e admits an expansion in this canonical basis:

$$O(X) \;=\; \sum_{i=0}^{n} a_i\, O_i(X), \quad a_i \in \mathbb{Z}_{p^e}.$$

Let $\mathcal{O}_{\text{mon}} \subseteq \mathcal{O}$ denote the set of monic null polynomials. For the canonical basis above, the p-power prefactor $p^{\max\{e-v_p(i!),0\}}$ equals 1 if and only if $v_p(i!) \geq e$, which is equivalent to the condition

$$i \;\geq\; \mu(p^e) \;:=\; \min\{\, m \in \mathbb{Z}_{>0} : v_p(m!) \geq e \,\}. \tag{3.4.1}$$

It follows that every monic null polynomial modulo p^e has degree at least $\mu(p^e)$. Since $p^e \mid (pe)!$, we have $\mu(p^e) \leq pe$.

Null Polynomials on a Restricted Residue Window [15] When plaintext values are known to lie within a smaller residue window, we can construct monic null polynomials of lower degree. Let w_i denote the i-th base-p digit of w ($i \geq 0$, with the least significant digit first), and let $w_{i,j}$ denote a value whose least-significant base-p digit equals w_i and whose next j base-p digits are zero ($j \geq 0$). Assume inputs satisfy

$$L \;\leq\; w_p\langle \Delta, e\rangle \;:=\; \sum_{i=0}^{\Delta-1} w_{i,\,e-1-i}\, p^i \;\leq\; H.$$

For any integer $l \geq 1$, we define

$$\Lambda_l(T) \;=\; \prod_{j=0}^{l-1} \left(\prod_{i=L}^{H} (T-i) \;-\; j \cdot p^{\,\Delta + v_p((H-L)!) - \lfloor \log_p(H-L) \rfloor} \right).$$

The polynomial $\Lambda_l(T)$ is monic and serves as a null polynomial over the restricted domain

$$\{w \in \mathbb{Z}_{p^e} : L \leq w_p\langle \Delta, e\rangle \leq H\} \subseteq \mathbb{Z}_{p^e},$$

provided that

$$e \;\leq\; l\big(\Delta + v_p\big((H-L)!\big) - \big\lfloor \log_p(H-L) \big\rfloor\big) + v_p(l!).$$

Equivalently, for a given target precision e, we select the smallest integer l satisfying the above inequality; under this choice, $\Lambda_l(w) \equiv 0 \pmod{p^e}$ holds for all admissible w. Furthermore,

$$\deg \Lambda_l(T) \;=\; l\,(H - L + 1), \tag{3.4.2}$$

and this degree (for the chosen l) does not exceed the global degree lower bound $\mu(p^e)$ derived from the falling-factorial construction.

When to Use the Null-Polynomial Optimization Given any target polynomial $f(X)$, we may select a monic null polynomial $O(X) \in \mathcal{O}_{\text{mon}}$ and form

$$f^\star(X) \;=\; f(X) + O(X). \tag{3.4.3}$$

By definition, $f^\star$ is pointwise equivalent to f over the relevant plaintext domain: $f^\star(t) = f(t)$ for all admissible t. The key benefit is that the degree of $f^\star$ can be adjusted to land on a decomposition-friendly degree—specifically, a multiple of the slot degree (e.g., $D^\star = k^\star d$)—by choosing O of an appropriate degree.

This optimization can increase the effective parameter $k^\star = D^\star/d$ due to the raised degree; however, it eliminates the need to handle an auxiliary remainder polynomial $a(X)$ as in Section 3.2. In many practical parameter regimes, the overall online cost is therefore lower after the degree adjustment, despite the larger $k^\star$. To decide when this optimization is beneficial, we use a lightweight

cost model and enable the optimization only when it reduces the ciphertext-multiplication count—a choice justified by the fact that both options share the same Galois orbit product (hence, they require the same automorphisms). The differentiating factor is the number of ciphertext multiplications spent on: (i) evaluating the sparse correction polynomial $g(t)$, and (ii) forming the base product term (e.g., $P(t)$) together with an additional low-degree remainder terms $a(t)$.

Let D_0 be the minimal-degree threshold for the existence of a monic null polynomial (cf. Eq. (3.4.1) and Eq. (3.4.2)). We round the degree of f to $D^\star := d\cdot\lceil D_0/d\rceil$, so $k^\star := \lceil D_0/d\rceil$. A key observation is that the components below reuse many of the same powers of $t \in \mathbb{Z}_{p^e}$. Therefore, instead of counting "baby-step" costs separately, we take the union of the required baby-step sets and charge the cost of materializing this union only once; we then add the component-specific multiplication overheads for assembling each component from these shared powers.

Global Baby-Step Union The components induce the following baby-step requirements:

- **Sparse** $g(t)$**.** Evaluating $g(t)$ requires the chain $\mathcal{B}_g = \{t^2, t^4, \ldots, t^{D_g}\}$.
- **Base product** $P(t)$**.** Under optimal-depth P-S with block size $s = \lfloor\sqrt{k}\rfloor$ ($\lfloor\sqrt{k^\star}\rfloor$ in the optimized setting), we require baby steps $\{t^2, \ldots, t^s\}$ together with the stride powers used by the doubling ladder: $\mathcal{B}_P = \{t^2, \ldots, t^s\} \cup \{t^{2s}, t^{4s}, \ldots\}$.
- **Remainder polynomial** $a(t)$ **(baseline only).** Evaluating $a(t)$ requires additional low-degree powers $\mathcal{B}_a \subseteq \{t^2, t^3, \ldots, t^{\lfloor\sqrt{\delta}\rfloor}\}$, where $\delta := D - kd \in \{0, 1, \ldots, d-1\}$, but no stride powers.

The final baby-step list is $\mathcal{B} = \mathcal{B}_g \cup \mathcal{B}_P$ (and additionally $\cup\, \mathcal{B}_a$ in the baseline). We materialize each distinct element of $\mathcal{B}$ once, and denote by BabyCost($\mathcal{B}$) the total number of ciphertext-ciphertext multiplications required to generate all powers in $\mathcal{B}$.

Component-Specific Overhead Beyond Baby Steps Once $\mathcal{B}$ is available, the remaining multiplication cost comes from assembling each component:

- **Base product** $P(t)$**.** After the required baby/stride powers for optimal-depth P-S are available in the global union $\mathcal{B}$, forming $P(t) = \prod_{i=1}^{k}(\alpha_i - t)$ incurs only the block-combination cost, with block size $s = \lfloor\sqrt{k}\rfloor$. We model this cost as $C_{\mathrm{prod}}(k) = \big(\lceil k/s\rceil - 1\big),\ s = \lfloor\sqrt{k}\rfloor$.($k^\star$ and $s^\star$in the optimized setting)
- **Remainder** $a(t)$ **(baseline only).** With block size $s_\delta = \lfloor\sqrt{\delta}\rfloor$, we model the extra overhead as $C_{\mathrm{rem}}(\delta) = \big(\lceil\delta/s_\delta\rceil - 1\big) + 1$, where the final "+1" accounts for the single ciphertext multiplication used to multiply $a(t)$ into the main product term (i.e., combining $a(t)$ with $\prod_i h_i(t)$).

Decision Rule Let $\mathcal{B}^{(\mathrm{base})}$ and $\mathcal{B}^{(\star)}$ be the final baby-step unions for the baseline and the null-polynomial optimized setting, respectively (the optimized setting omits $\mathcal{B}_a$ but uses $k^\star$). We compare the two options by their total multiplication counts:

$$\mathrm{Cost}_{\mathrm{base}} := \mathrm{BabyCost}\big(\mathcal{B}^{(\mathrm{base})}\big) \;+\; C_{\mathrm{prod}}(k) \;+\; C_{\mathrm{rem}}(\delta),$$

$$\mathrm{Cost}_{\star} := \mathrm{BabyCost}\big(\mathcal{B}^{(\star)}\big) \;+\; C_{\mathrm{prod}}(k^\star),$$

and enable the null-polynomial optimization if and only if $\mathrm{Cost}_\star < \mathrm{Cost}_{\mathrm{base}}$.

4 Applying to Digit Extraction in BGV Bootstrapping

We now turn to the flagship application of our algorithm: accelerating the digit-extraction stage in BGV bootstrapping. Bootstrapping is indispensable for enabling fully homomorphic evaluation, yet it remains one of the most computationally expensive components in practical deployments—with digit extraction typically constituting its dominant bottleneck. By substantially reducing the cost of evaluating digit-extraction polynomials, our algorithm yields notable improvements in the overall efficiency of BGV bootstrapping. In this section, we first review the standard digit-extraction workflow, then elaborate on why its structural properties align particularly well with the evaluation regime of our algorithm, and finally present an end-to-end performance comparison against state-of-the-art BGV bootstrapping implementations.

4.1 Digit Extraction in BGV Bootstrapping

Digit extraction is the polynomial evaluation subroutine that enables modulus reduction in BGV bootstrapping. Informally, given a ciphertext encrypting a plaintext $w \in \mathbb{Z}_{p^{r+\Delta}}$(i.e.,$e = r + \Delta$), the goal is to homomorphically recover the Δ least significant base-p digits of w—this emulates the p-adic truncation that occurs during homomorphic decryption. The modern paradigm for this technique originates from Halevi and Shoup's (HS) original BGV bootstrapping framework, which introduced practical p-adic digit extraction as the linchpin of modulus reduction in BGV [13]. Subsequent works have refined both the extraction polynomials and their evaluation strategies—most notably those by Chen and Han [4], Geelen *et al.* (often referred to as GIKV) [9], and Ma *et al.* (MHW) [15]—yielding extraction-polynomials of lower degree and more computationally efficient evaluation in practice.

Write the base-p expansion of w as

$$w = \sum_{i \geq 0} w_i p^i, \quad w_i \in \{0, 1, \ldots, p-1\}.$$

Following the standard literature on p-adic bootstrapping, we adopt the notation $w_{i,j} \in \mathbb{Z}_{p^e}$ for a value that matches the i-th base-p digit of w modulo p and is lifted to a precision of p^{j+1}, satisfying

$$w_{i,j} \;\equiv\; w_i \pmod{p^{j+1}}, \quad 0 \leq j \leq e-1.$$

Equivalently, $w_{i,j}$ can be written as $w_i + p^{j+1}q$ for some integer q—in other words, the higher base-p digits beyond position j are irrelevant for our analysis. For a target window length Δ, digit extraction seeks to recover the Δ least significant base-p digits of w. In its canonical form, the desired output is given by

$$w_p\langle \Delta, e\rangle = \sum_{i=0}^{\Delta-1} w_{i,e-1-i} \cdot p^i. \tag{4.1.1}$$

We next introduce two polynomials central to the digit extraction procedure, and we use these as core components to elaborate on the full digit extraction workflow:

Lemma 6 (Lifting Polynomial [13]). *For any $k \in \{1, \ldots, e-1\}$, a* lifting polynomial *is a polynomial $L_k \in \mathbb{Z}_{p^e}[X]$ of degree p such that for all admissible intermediate values $w_{i,j-1} \in \mathbb{Z}_{p^e}$ with $j \leq k$,*

$$L_k\big(w_{i,j-1}\big) \;=\; w_{i,j}. \tag{4.1.2}$$

We can retrieve the i-th base-p digit w_i in the form of $w_{i,0}$, which is obtained via $\mathbb{Z}_{p^e}$-linear combinations of w and the previously recovered digits. For instance, one convenient recurrence relation is given by

$$w_{i,0} \;=\; \frac{w - \sum_{j=0}^{i-1} w_{j,\,i-j}\, p^j}{p^i}. \tag{4.1.3}$$

An alternative—and often more computationally efficient—approach bypasses the iterative lifting chain by employing a single *magic polynomial* that directly extracts the least significant base-p digit at full precision.

Lemma 7 (Magic Polynomial [4]). *For $e \geq 2$, the* magic polynomial *$G_e(X) \in \mathbb{Z}_{p^e}[X]$ is a polynomial of degree $(e-1)(p-1)+1$ such that for all $w \in \mathbb{Z}_{p^e}$,*

$$G_e(w) \;\equiv\; w_0 \pmod{p^e}. \tag{4.1.4}$$

We present an example of digit extraction with $e = 6$ and $\Delta = 2$ (see the figure below): starting from the input w, we first derive $w_{0,1}$ via the lifting polynomial and combine it with w to form $w_{1,0}$; we then compute $w_{0,5}$ and $w_{1,4}$ from $w_{0,0}$ and $w_{1,0}$ using the magic polynomial. The final desired result is obtained via $w_p\langle 2, 6\rangle = w_{0,5} + w_{1,4} \cdot p$, which corresponds to the 2 least significant base-p digits of w.

4.2 Accelerating Digit Extraction via Our Algorithm

Prior work has focused on optimizing the degree of digit-extraction polynomials. In contrast, our algorithm targets a distinct dimension: it enables a more computationally efficient evaluation strategy for a given polynomial. As such, our primitive complements these advances in polynomial construction—once a

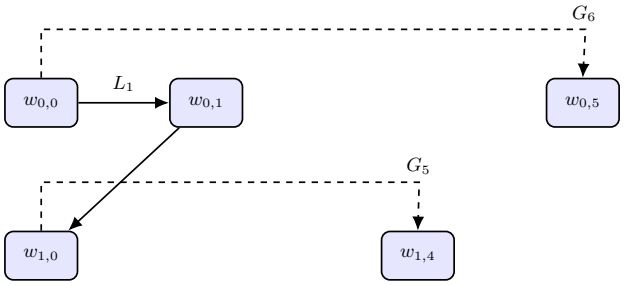


Fig. 2. Digit extraction dataflow using lifting polynomials L_k and magic polynomials G_e.

digit-extraction polynomial is fixed, our approach reduces the ciphertext multiplication cost of its evaluation.

Two key features of modern digit-extraction designs make them particularly well suited to the evaluation regime of our algorithm:

(1) Null-Polynomial Degree Tightening Aligns with Our Null-Polynomial Optimization. Recent implementations optimize the degree of magic polynomials using monic null polynomials [9, 15]. This places the optimized extraction polynomial within the parameter range where our null-polynomial optimization (Section 3.4) is most effective: the rounded degree $D^\star = k^\star d$ typically satisfies $k^\star \leq k+1$ (at most an increment of 1), while eliminating the need to evaluate a non-trivial auxiliary remainder polynomial $a(X)$. In other words, a modest increase in degree can substantially simplify the instantiation of our algorithm and reduce the overall computational cost.

(2) Small Input Domain from Sparse-Key Digit Extraction Enables Lower-Degree Null Polynomials. The sparse-key digit-extraction variant proposed in [15] restricts the plaintext input domain to a narrow residue window. This allows for the use of restricted-domain null polynomials from Section 3.4—rather than global null polynomials—yielding significantly lower degrees. Consequently, across practical BGV parameter sets, the resulting digit-extraction polynomials can be maintained within the degree range where our algorithm achieves logarithmic multiplicative complexity.

4.3 Implementation

Baseline. We implement our algorithm-based bootstrapping routine in HElib, following the parameter sets reported in [15], and compare it against the bootstrapping implementation from [15]—which, to the best of our knowledge, represents the fastest publicly available open-source BGV bootstrapping configuration in HElib at the time of writing.

For each parameter set, our method and the baseline use identical cryptographic parameters (including $p, r, \Delta, M, \varphi(M)$, and the slot degree d), with all

bootstrapping components held constant except for the digit-extraction subroutine.

Case	p	r	Δ	M	$\varphi(M)$	d
I	17	4	2	38309	36960	24
II	127	2	1	56647	55080	45
III	257	2	1	55427	54096	28
IV	8191	1	1	45193	44100	14
V	65537	1	1	50731	50112	18

Table 3. BGV parameter sets used in our evaluation (matching the configuration in [15]).

Experimental Setup. All experiments were conducted on a single machine equipped with an AMD Ryzen 9 9950X3D CPU and 64 GB of RAM. We use HElib commit `3e337a66a91a` (NTL version `11.5.1-1`, GMP version `2:6.2.1+dfsg-3ubuntu1`). The operating system deployed was `Ubuntu 22.04.3 LTS`. For each parameter set, we repeated the full bootstrapping procedure over $N = 100$ independent trials and reported the average wall-clock time (and standard deviation) for the digit extraction stage and the overall time. Timing measurements were collected using `std::chrono::steady_clock` (wall-clock time).

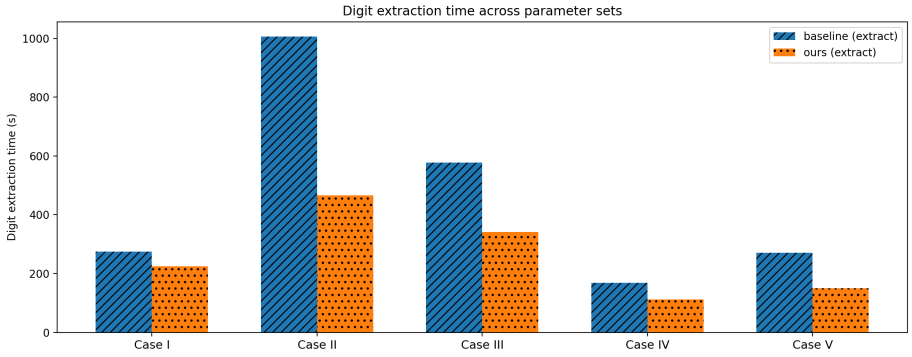


Fig. 3. Digit-extraction time for each parameter set. The left bar is the baseline [15] and the right bar is ours.

Implementation Results. Across all five parameter sets in Table 2, our method consistently reduces the runtime of digit extraction, achieving a digit-extraction speedup ranging from 1.22× to 2.16×. This reduction translates to an end-to-

Case	Total (s)			Digit extraction (s)			cap/time		
	Baseline	Ours	Speed-up	Baseline	Ours	Speed-up	Baseline	Ours	Impr.
I	314.55	262.31	1.20	274.48	225.04	1.22	1.38	1.66	1.20
II	1286.02	821.22	1.57	1005.94	465.39	2.16	0.84	1.27	1.51
III	1607.31	1231.93	1.30	577.36	341.08	1.69	0.62	0.78	1.26
IV	211.98	156.33	1.36	168.52	112.31	1.50	3.47	4.23	1.22
V	318.94	197.88	1.61	270.59	149.96	1.80	2.56	3.62	1.41

Table 4. Bootstrapping runtime and digit-extraction breakdown. The baseline is the implementation from [15]. All numbers are averaged over $N = 100$ trials.

end bootstrapping speedup of 1.20× to 1.61× (depending on the parameter regime)—as reflected by the Speed-up metric in Table 4.

For the configuration with $(p = 127, r = 2)$, the digit-extraction runtime is reduced from 1005.94 s to 465.39 s (a 2.16× speedup), and its proportion of the overall bootstrapping time drops substantially (from approximately 78% to 57%). In contrast, for the configuration with $(p = 257, r = 2)$, the combined runtime of the linear transforms (**linear1**/**linear2**) reaches as high as 1000 s—such that the speedup achieved in digit extraction does not translate to a statistically significant improvement in the end-to-end bootstrapping runtime. This indicates a bottleneck shift in some parameter regimes: after accelerating digit extraction, the linear transforms (**linear1**/**linear2**) become comparatively more dominant in the overall runtime. Consequently, further end-to-end performance gains will increasingly depend on optimizing key-switching and linear-algebra kernels, rather than polynomial evaluation alone.

The reason our algorithm yields less pronounced improvements for the parameter set $(p = 17, r = 4)$ is as follows: digit extraction under this parameter set involves two rows (cf. Fig. 2), requiring us to evaluate three polynomials with degrees $D = 15$, 17, and 81, respectively. Our algorithm only achieves a significant advantage when evaluating the polynomial of degree $D = 81$—for $D = 15$ and 17, $\log D$ is nearly equal to $\sqrt{D}$ in magnitude, and thus our complexity bound (Proposition 2) does not outperform that of the Paterson–Stockmeyer (P-S) algorithm by a meaningful margin.

Noise Budget Metric. Beyond runtime, we report the capacity (in bits) consumed during the digit extraction phase— which depends on the multiplicative depth of the algorithm—in Table 5, as well as an efficiency metric (defined as the ratio of the remaining noise budget after bootstrapping to the bootstrapping wall-clock time, denoted cap/time) in Table 4; this metric quantifies the number of noise budget bits delivered per unit runtime. In our instantiation, the multiplicative depth of digit extraction may increase by 1–2 levels for certain slot degrees d with non-smooth factorizations (e.g., those containing prime factors 5 or 7), owing to suboptimal choices in Galois orbit decomposition (cf. Section 3.1); an additional multiplicative depth level also arises when combining $a(t)$ with $\prod_{i=1}^{k} h_i(t)$ in cases where $D \neq kd$. This results in our algorithm consuming

an additional 37.43–108.06 bits of capacity during digit extraction; nevertheless, the improvement in cap/time remains nearly on par with the end-to-end bootstrapping speedup. This indicates that while our algorithm may induce a slight increase in multiplicative depth for some parameter sets, the resulting efficiency loss is negligible—and the majority of the latency reduction translates to proportional gains in cap/time.

Case	Mult. depth		Capacity consumption (bits)		Ours/Baseline
	Baseline	Ours	Baseline	Ours	
I	9	9	340.35	340.18	0.9995
II	8	9	290.64	328.07	1.1288
III	8	9	315.80	352.91	1.1175
IV	7	9	287.76	374.26	1.3006
V	7	9	331.40	439.46	1.3261

Table 5. Multiplicative depth and capacity consumption compared to the baseline [15].

Instantiation Choice and Offline Cost Tradeoffs. For the $(p = 8191, r = 1)$ and $(p = 65537, r = 1)$ configurations, we do *not* apply the null-polynomial optimization (Section 3.4). Instead, we instantiate our algorithm using the general decomposition described in Equation (3.2.1), namely:

$$f(X) = a(X) \cdot H(X) + g(X),$$

and evaluate the low-degree remainder polynomial $a(X)$ as part of the online procedure.

In principle, applying the null-polynomial optimization would allow us to replace f with a pointwise-equivalent polynomial $f^\star = f + O$ whose degree is rounded to a multiple of d compatible with our algorithm. This eliminates the need to handle $a(X)$ and reduces the online computational cost—under our cost model, this reduction corresponds to two fewer ciphertext multiplications and one fewer level of multiplicative depth for both parameter sets. However, for these parameter sets, this more aggressive instantiation would entail substantially heavier offline computation in the decomposition search: after degree rounding, the effective quotient $k^\star = \lfloor \deg(f^\star)/d \rfloor$ is relatively large ($k^\star = 6$ for $(p = 65537, r = 1)$ and $k^\star = 7$ for $(p = 8191, r = 1)$). Consequently, in our current implementation, the compile-time search for a valid correction polynomial g is computationally intensive to the point of being infeasible on a personal computer. We therefore omit the null-polynomial optimization for these two configurations, and frame this optimization as an optional offline enhancement for future work or for deployments with more powerful offline computing resources.

5 Conclusion and Future Work

We introduce faster polynomial evaluation algorithms for SIMD-style RLWE-based FHE schemes that leverages Galois-orbit products alongside sparse corrections to reduce the ciphertext-ciphertext multiplicative cost across a broad range of polynomial degrees. We instantiated our algorithm in BGV/HElib and demonstrated that accelerating digit extraction yields substantial end-to-end bootstrapping speedups.

Several directions remain for future work. First, a full algebraic proof for the existence of decompositions for the power-of-two sparse family would strengthen our theoretical foundations beyond the current heuristic and empirical evidence. Second, improving the compile-time search for sparse correction polynomials—i.e., developing more efficient algorithms for finding g—could expand the practical applicability of our algorithms. Third, we aim to further modify the plaintext SIMD structure of RLWE-based FHE schemes to enable polynomial evaluation to achieve $O(\log D)$ ciphertext multiplications for arbitrary degrees D across a wider range of instantiations. Finally, we seek to extend the applicability of our algorithm from the constraint $D \leq d \log p$ to polynomials of arbitrary degree, thereby enabling its use in a broader range of application scenarios that require the computation of high-degree univariate polynomials.

Acknowledgments

We thank the anonymous reviewers of CRYPTO 2026 for their helpful comments and suggestions on earlier versions of our paper. This work is supported by the National Natural Science Foundation of China (Grant Nos. 62525205 and 62372447), the National Key Research and Development Program of China (Grant No. 2023YFB4503203) and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0690200).

References

1. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS). pp. 309–325. ACM (2012). https://doi.org/10.1145/2090236.2090262
2. Chebolu, S.K., Mináč, J.: Counting irreducible polynomials over finite fields using the inclusion-exclusion principle. Mathematics Magazine **84**(5), 369–371 (2011), also available as arXiv:1001.0409
3. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology – EUROCRYPT 2019. LNCS, vol. 11477, pp. 34–54. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_2
4. Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. Lecture Notes in Computer Science, vol. 10820, pp. 315–337. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_12

5. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology – EUROCRYPT 2018. LNCS, vol. 10820, pp. 360–384. Springer (2018). https://doi.org/10.1007/978-3-319-78381-9_14
6. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology – ASIACRYPT 2017. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_15
7. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144
8. von zur Gathen, J., Shoup, V.: Computing frobenius maps and factoring polynomials. Computational Complexity **2**, 187–224 (1992). https://doi.org/10.1007/BF01272074
9. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14006, pp. 257–286. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30620-4_9
10. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC). pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440
11. Halevi, S., Shoup, V.: Design and implementation of a homomorphic-encryption library. Tech. rep., IBM Research (2013), https://people.csail.mit.edu/shaih/pubs/he-library.pdf, hElib technical report
12. Halevi, S., Shoup, V.: HElib design principles. Manuscript (Aug 2020), version 0.1
13. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(7) (2021). https://doi.org/10.1007/s00145-020-09368-7
14. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Topics in Cryptology – CT–RSA 2020. LNCS, vol. 12006, pp. 364–390. Springer (2020). https://doi.org/10.1007/978-3-030-40186-3_16
15. Ma, S., Huang, T., Wang, A., Wang, X.: Accelerating BGV bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. Lecture Notes in Computer Science, vol. 14652, pp. 403–432. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58723-8_14
16. Okada, H., Player, R., Pohmann, S.: Homomorphic polynomial evaluation using galois structure and applications to BFV bootstrapping. In: Advances in Cryptology - ASIACRYPT 2023. Lecture Notes in Computer Science, vol. 14440, pp. 640–671. Springer (2023). https://doi.org/10.1007/978-981-99-8728-1_22
17. Paterson, M.S., Stockmeyer, L.J.: On the number of multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973)
18. Peikert, C., Pepin, Z.: Vive galois! part 1: Optimal simd packing and?packed bootstrapping for?fhe. In: Applebaum, B., Lin, H.R. (eds.) Theory of Cryptography. pp. 185–219. Springer Nature Switzerland, Cham (2026)

A Additional proofs

Proposition 4 (Cost Bound of Evaluating β). *Let $S_m = 1 + X + \cdots + X^m$ and $P_m = X^m$. For any positive integer D, the multiplicative complexity*

$M(D)$ of computing the pair (S_D, P_D) is at most $3\lfloor \log D \rfloor$. Consequently, the multiplicative complexity of computing S_m is bounded by $3\lfloor \log D \rfloor$. Additionally, the multiplicative depth $N(m)$ of S_m is at most $2\lfloor \log D \rfloor$.

Proof. We maintain two quantities simultaneously:

- $P_m := X^m$,
- $S_m := 1 + X + \cdots + X^m$.

Case 1: m to $2m$ (even) We have

$$S_{2m} = S_m + \underbrace{(X^{m+1} + \cdots + X^{2m})}_{=X^m(X+\cdots+X^m)=X^m(S_m-1)},$$

which simplifies to

$$S_{2m} = S_m + P_m \cdot (S_m - 1).$$

For the monomial, we have

$$P_{2m} = P_m^2.$$

Thus, computing (P_{2m}, S_{2m}) from (P_m, S_m) requires two multiplications:

1. P_m^2 (one multiplication),
2. $P_m \cdot (S_m - 1)$ (one multiplication).

Therefore, the multiplicative complexity satisfies the recurrence

$$M(2m) = M(m) + 2.$$

Case 2: m to $2m+1$ (odd) We first compute up to $2m$, then add the final term:

$$P_{2m+1} = P_{2m} \cdot X, \quad S_{2m+1} = S_{2m} + P_{2m+1}.$$

This corresponds to one additional multiplication beyond the even case, leading to

$$M(2m + 1) = M(m) + 3.$$

Closed-form complexity: Let $D \geq 1$ and write it in binary. Define $\mathrm{wt}(D)$ as the Hamming weight of D (the number of 1s in its binary representation). The recurrence shifts right by one bit at each step, contributing $+2$ per level, with an extra $+1$ for each least significant bit that is 1. With the base case $M(1) = 0$, we obtain

$$M(D) = 2\lfloor \log D \rfloor + \mathrm{wt}(D) - 1.$$

Since $\mathrm{wt}(D) \leq \lfloor \log D \rfloor + 1$, we have

$$M(D) \leq 3\lfloor \log D \rfloor,$$

as required.

Multiplicative depth: As the recurrence shifts right by one bit, the depth increases by 1 per level, with an additional +1 for each least significant bit that is 1. With the base case $N(1) = 0$, we obtain

$$N(D) = \lfloor \log D \rfloor + \mathrm{wt}(D) - 1 \leq 2\lfloor \log D \rfloor,$$

which completes the proof.