Accelerating BGV Bootstrapping for Large p Using Null Polynomials Over $\mathbb{Z}_{p^e}$

Shihe Ma[1], Tairong Huang[2], Anyu Wang[2,3,4(✉)], and Xiaoyun Wang[2,3,4,5,6]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, msh21@mails.tsinghua.edu.cn
[2] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China, htr19@mails.tsinghua.edu.cn, anyuwang,xiaoyunwang@tsinghua.edu.cn
[3] Zhongguancun Laboratory, Beijing, China
[4] National Financial Cryptography Research Center, Beijing, China
[5] Shandong Institute of Blockchain, Jinan, China
[6] Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), School of Cyber Science and Technology, Shandong University, China

Abstract. The BGV scheme is one of the most popular FHE schemes for computing homomorphic integer arithmetic. The bootstrapping technique of BGV is necessary to evaluate arbitrarily deep circuits homomorphically. However, the BGV bootstrapping performs poorly for large plaintext prime p due to its digit removal procedure exhibiting a computational complexity of at least $O(\sqrt{p})$. In this paper, we propose optimizations for the digit removal procedure with large p by leveraging the properties of null polynomials over the ring $\mathbb{Z}_{p^e}$. Specifically, we demonstrate that it is possible to construct low-degree null polynomials based on two observations of the input to the digit removal procedure: 1) the support size of the input can be upper-bounded by $(2B+1)^2$; 2) the size of the lower digits to be removed can be upper-bounded by B. Here B can be controlled within a narrow interval $[22, 23]$ in our parameter selection, making the degree of these null polynomials much smaller than p for large values of p. These low-degree null polynomials can significantly reduce the polynomial degrees during homomorphic digit removal, thereby decreasing both running time and capacity consumption. Theoretically, our optimizations reduce the computational cost of extracting a single digit from $O(\sqrt{pe})$ (by Chen and Han) or $O(\sqrt{p}\sqrt[4]{e})$ (by Geelen et al.) to $\min(2B+1, \sqrt{\lceil e/t \rceil (2B+1)})$ for some $t \geq 1$. We implement and benchmark our method on HElib with $p = 17, 127, 257, 8191$ and 65537[1]. With our optimized digit removal, we achieve a bootstrapping throughput $1.38 \sim 151$ times that in HElib, with the speedup increasing with the value of p. For $p = 65537$, we accelerate the digit removal step by 80 times and reduce the bootstrapping time from more than 12 hours to less than 14 minutes.

Keywords: BGV · Bootstrapping · FHE · Homomorphic Digit Removal · Null Polynomial

[1] The code is available at https://github.com/msh086/BGV-Boot-for-Large-p.

1 Introduction

Since Gentry's first construction of the Fully Homomorphic Encryption (FHE) scheme in 2009 [10], research in FHE has seen a surge following Gentry's blueprint. Among the various FHE schemes, Brakerski-Gentry-Vaikuntanathan (BGV) [5] and Brakerski-Fan-Vercauteren (BFV) [7] are the most popular schemes for homomorphic integer arithmetic. FHE relies on bootstrapping to remove the noise accumulated during homomorphic operations to enable infinite homomorphic computation on ciphertexts. The bootstrapping procedure developed by Halevi and Shoup [13,15] and implemented in HElib [16] is the foundational blueprint for all subsequent works and improvements [6,8,9]. However, BGV bootstrapping is highly time-consuming and is the bottleneck for evaluating deep homomorphic circuits.

In a nutshell, BGV bootstrapping, like bootstrapping in other FHE schemes, computes the decryption function homomorphically. Most existing methods for BGV bootstrapping (including the works of Halevi-Shoup [13,15] and Geelen-Vercauteren [9]) require homomorphically removing the lower digits of encrypted p-radix values, which corresponds to the noise-removal operation in BGV decryption. Specifically, given $p^{e-r}m + \epsilon \in \mathbb{Z}_{p^e}$, the digit removal procedure computes $m \in \mathbb{Z}_{p^r}$ by homomorphically evaluating a series of polynomials. In contrast, Kim et al. designed a bootstrapping framework without digit removal [17]. They first obtain a noiseless encryption of ϵ under a small ciphertext modulus Δ and an encryption of $m + \epsilon \in \mathbb{Z}_{p^r}$. Then, they use the TFHE functional bootstrap to obtain an encryption of ϵ under a large modulus, where the dimension and number of functional bootstraps equal the ring dimension. Finally, m is obtained by subtracting ϵ from $m + \epsilon$. However, their method has a computational cost growing at least quadratically with the ring dimension, thus requiring a small ring dimension for efficiency. With either digit removal or functional bootstrap, removing ϵ is the most computationally heavy part of BGV bootstrapping.

When the digit removal is performed modulo p^e for some prime p, the number of levels and computational cost to extract a single digit is $(e-1)\log_2(p)$ and $2(e-1)\sqrt{p}$ respectively for the algorithm in [13]. If the Chen-Han optimization is applied [6], the depth and the computational cost would be $\log_2((p-1)(e-1)+1)$ and $2\sqrt{(e-1)(p-1)+1}$ instead. The most recent progress on BGV bootstrapping includes the works of Geelen et al. [8,9]. In their first paper, Geelen et al. exploit the null polynomials modulo p^e and other optimizations to reduce the computational complexity of homomorphic digit removal. Asymptotically, the computational cost of extracting a single digit is reduced from $O(\sqrt{pe})$ (using Chen-Han's method) to $O(\sqrt{p}\sqrt[4]{e})$, at the cost of consuming $\log_2(p)$ more levels. In the second paper, Geelen and Vercauteren design a new approach to simplify the decryption formula evaluated by BGV bootstrapping, which unifies the bootstrapping of BGV and BFV under a single framework.

Our Contributions. Despite all the optimizations on BGV bootstrapping, its cost still *"scales very poorly with large p"*, as commented by Halevi in issue #80

of HElib [12]. This is because the polynomials evaluated during digit removal have degrees that are at least p, implying a complexity of at least $O(\sqrt{p})$ when evaluating these polynomials. This may explain why HElib and other works choose small p for BGV bootstrapping, e.g., p up to 127 or 257. In contrast, large p's are favored to perform high-precision integer arithmetic in practice, and one of the common choices is $p = 65537 = 2^{16} + 1$.

In this work, we address the open problem of bootstrapping with large values of p by introducing novel optimizations for the homomorphic digit removal process. Our contributions are summarized as follows.

(1) We propose two new constructions of low-degree null polynomials over $\mathbb{Z}_{p^e}$ to reduce the degree of the polynomials involved in homomorphic digit removal.

- The first construction exploits the sparsity of the input for digit removal and gives null polynomials with a maximum degree of $(2B+1)^2$. Here, B represents the bound of the coefficients of $\lfloor \epsilon_0 + \epsilon_1 s \rceil$, where ϵ_i's are random polynomials whose coefficients are sampled uniformly from $(-1/2, 1/2)$ and s is the secret key with Hamming weight h. When the cyclotomic order M has only a few prime factors, B can be approximated as being proportional to $\sqrt{h}$. In all of our selected parameters, the actual value of B can be controlled within a narrow interval $[22, 23]$. Meanwhile, the value of p is distributed within the interval $[17, 65537]$, and the value of p^r ranges from 13 to 16 bits. The explicit formula for B is quite complex, so we defer it to Lemma 6.
- The second construction leverages the fact that the lower digits in digit removal are bounded by B, and gives null polynomials with degree upper bounded by $\lceil e/t \rceil (2B+1)$ for some integer t satisfying $t \geq \log_p(2B+1)$ and $e - t = r > 0$, where r is determined by the plaintext modulus p^r. We note that the bounded property of the lower digits has been utilized by Geelen et al. for the construction of null polynomials [8]. However, in their work, the digit extraction polynomial is expressed as the composition of multiple sub-polynomials, and null polynomials are employed to accelerate the evaluation of these sub-polynomials. In contrast, we demonstrate that it is possible to directly use the null polynomials to accelerate the evaluation of the digit extraction polynomial, without relying on function composition.

By employing these two types of null polynomials, we can reduce the computational complexity of extracting a single digit to $\min(2B+1, \sqrt{\lceil e/t \rceil (2B+1)})$. In contrast, previous methods require a complexity of either $O(\sqrt{pe})$ [6] or $O(\sqrt{p}\sqrt[4]{e})$ [8]. Table 1 compares the depth and computing cost of digit extraction in previous and our works. Since we optimize bootstrapping by reducing the digit extraction polynomial modulo a low-degree null polynomial, both the depth and the computing cost can be saved as long as the null polynomial has a degree smaller than the original polynomial.

(2) We implement our optimized digit removal on HElib and benchmark it with the HS bootstrapping implemented in HElib as the baseline. For $p = 17, 127, 257, 8191$ and 65537, our method has a throughput 1.38x$\sim$151x that of

HS bootstrapping. Specifically, for $p = 65537$, we reduce the running time of the digit removal step by 80 times and obtain a 151x throughput.

Table 1: Depth and computing cost of the digit removal procedure in modulo $\mathbb{Z}_{p^e}$, both in previous and our works. The number of non-scalar multiplications is estimated assuming odd BSGS optimization [8].

Method	No. of non-scalar mult	Depth consumption
CH18 [6]	$\sqrt{2((p-1)(e-1)+1)}$	$\log_2((p-1)(e-1)+1)$
GIKV23 [8]	$2\sqrt{2p}\sqrt[4]{e}$	$2\log_2(p)+\log_2(e)$
Ours(first type)	$\sqrt{2\lceil\frac{e}{t}\rceil(2B+1)}$	$\log_2(2B+1)+\log_2(\lceil\frac{e}{t}\rceil)$
Ours(second type)	$\sqrt{2}(2B+1)$	$2\log_2(2B+1)$

Our Techniques. The digit removal procedure in BGV bootstrapping, as suggested in [13,15,9], aims to extract the highest r digits of $p^{e-r}m+\epsilon \in \mathbb{Z}_{p^e}$ for plaintext $m \in \mathbb{Z}_{p^r}$ and noise ϵ, which corresponds to the rounding step of the decryption function. This procedure is currently realized with digit removal polynomials with degrees at least p [13,6,8]. Our direction of optimization is to exploit the null polynomials over $\mathbb{Z}_{p^e}$ to reduce the degree of digit removal polynomials, thereby reducing both the depth and computating cost of digit removal. Although it has been observed by Geelen et al. [8] that low degree null polynomials can be constructed over $\mathbb{Z}_{p^e}$ by utilizing the bounded property of the lower digits, we demonstrate that this kind of polynomials can be employed to expedite the evaluation of digit removal polynomial in a more straightforward and effective manner.

To facilitate our optimization, we adapt the digit removal process with the aim of computing $\epsilon \in \mathbb{Z}_{p^e}$ from $\epsilon^*\Delta+\epsilon \in \mathbb{Z}_{p^e}$ by employing the techniques proposed by Kim et al. [17], where Δ is an integer and ϵ^*, ϵ have a small bound B. By doing this, we can construct low-degree polynomials to assist in the digit removal process from the following two perspectives.

Firstly, we observe that for such digit removal, the input $\epsilon^*\Delta+\epsilon$ has a small support size of $(2B+1)^2$, which is independent of the message or noise in the input ciphertext. Consequently, we can construct a *global null polynomial* $\prod_{a\in\mathrm{supp}(\epsilon^*\Delta+\epsilon)}(X-a)$, which is a monic null polynomial of degree $(2B+1)^2$ over $\mathrm{supp}(\epsilon^*\Delta+\epsilon)$. Given that B is typically small (around $20 \sim 30$ in practice), the degree of this null polynomial can be smaller than p when p is large. In contrast, in HS or GV bootstrapping, the support size of $p^{e-r}m+\epsilon$ can be as large as p^e, making the construction of such low-degree null polynomials infeasible.

Secondly, we observe that the lower digits ϵ have smaller sizes than that in HS or GV bootstrapping when $\Delta = p^t$ for some integer t. Based on this, we can construct monic *local null polynomials* of degree no more than $\lceil e/t\rceil(2B+1)$ over $\mathrm{supp}(\epsilon^*\Delta+\epsilon)$, which have the form of $\prod_{j=0}^{\lceil e/t\rceil-1}(\prod_{i\in\mathrm{supp}(\epsilon)}(X-i)-j\cdot p^v)$ for

some $v \geq t$. Although the degree of this polynomial now depends on e, its actual value is typically much smaller than $(2B+1)^2$ since $\lceil e/t \rceil$ is usually smaller than $2B+1$, particularly for large p.

Utilizing these two types of low-degree null polynomials, we can effectively reduce the polynomials that need to be evaluated during the digit removal process, producing polynomials with the same effects but much lower degrees. This reduction directly decreases the depth and computing cost of evaluating these polynomials and the BGV bootstrapping. Furthermore, given that B is proportional to $\sqrt{h}$, we can achieve a smaller B by decreasing h using the sparse bootstrapping key encapsulation technique developed by Bossuat et al. [4], which in turn significantly reduces the degree of digit removal polynomials.

Finally, in contrast to the functional bootstrap approach by Kim et al. [17], our approach employs digit removal to compute $\mathrm{Enc}(\epsilon; p^{r+v_p(\Delta)})$. This computation inherently leverages the SIMD acceleration in BGV, leading to a substantial increase in efficiency. This benefit can be especially pronounced in scenarios involving large ring dimensions.

Roadmap. Section 2 defines the necessary notations and provides some background knowledge related to this paper, including an overview of BGV bootstrapping and a brief introduction to some mathematical concepts. Section 3 describes the digit removal procedure and existing optimizations on it. Section 4 gives more technical details and related proofs of our optimization on digit removal. Section 5 compares our results to previous results by an implementation based on HElib [16].

2 Preliminary

2.1 Basic Notations

- For any $M \in \mathbb{Z}^+$, we use $\Phi_M(X)$ to represent the M-th cyclotomic polynomial, $\phi(M)$ to represent Euler's totient function. We denote $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for any positive integer q.
- For any $q \in \mathbb{Z}^+$, we denote the set of representatives of $\mathbb{Z}_q$ as $[\![q]\!]$, which is $\{0,1\}$ if $q=2$, and $[-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q-1}{2} \rfloor]$ if $q > 2$. For $a \in \mathbb{Z}$, we use $[a]_q \in [\![q]\!]$ to denote its value modulo q. We say that a is q-reduced if $a = [a]_q$.
- For $q = p^e$, where p is an odd prime[7], any integer $a \in [\![q]\!]$ can be represented as a p-radix integer with e digits, i.e., $a = \sum_{i=0}^{e-1} p^i \cdot a_i$, where each digit $a_i \in [\![p]\!]$. We denote the p-radix digits of a as $a\langle i \rangle_p$, and the integer formed by the digits $a\langle i \rangle_p, a\langle i-1 \rangle_p, \ldots, a\langle j \rangle_p$ as $a\langle i,j \rangle_p = \sum_{k=j}^{i} p^{k-j} \cdot a\langle k \rangle_p \in [\![p^{i-j+1}]\!]$. For a polynomial $m \in \mathcal{R}_{p^e}$, represented under a specific polynomial basis, we extend the above notations to m by applying them to its coefficients.

[7] For $p=2$, $a \in [\![2^e]\!]$ can be similarly represented using the two's complement encoding. However, we omit the details since we will not deal with $p=2$ in this paper.

– A list or vector of n values, $a_0, \ldots, a_{n-1}$, is represented as $[a_0, a_1, \ldots, a_{n-1}]$. The infinity norm of a vector v is denoted by $|v|$. For a polynomial $a \in \mathcal{R}$, the infinity norm of the coefficient vector of a is denoted by $|a|$, provided that the polynomial basis is clear from the context.

– The support of a random variable a is denoted as $\mathrm{supp}(a)$. We sometimes exclude from $\mathrm{supp}(a)$ the low-probability tails of a (e.g., tails with probability less than 2^{-32}).

2.2 Canonical and Powerful Norms

Let $\omega = e^{2\pi i/M} \in \mathbb{C}$ be an M-th primitive root of unity. The canonical embedding is a homomorphism from $\mathcal{R}$ to $\mathbb{C}^{\phi(M)}$, defined as

$$\mathrm{canon}(a) = [a(\omega^i)]_{i \in \mathbb{Z}_M^*}, \quad \forall a \in \mathcal{R}. \tag{1}$$

The *canonical norm* is then defined as $|a|^{\mathrm{can}} = |\mathrm{canon}(a)| \in \mathbb{R}$.

Another norm commonly used in FHE schemes is the powerful norm. Assume $M = \prod_{i=1}^{k} M_i$, where the M_i's are pairwise coprime. Then there is an isomorphism between $A = \mathbb{Z}[X_1, \ldots, X_k]/(\Phi_{M_1}(X_1), \ldots, \Phi_{M_k}(X_k))$ and $\mathcal{R}$, namely

$$f(X_1, \ldots, X_k) \to f(X^{M/M_1}, \ldots, X^{M/M_k}). \tag{2}$$

The powerful representation of an element $a \in \mathcal{R}$ is denoted as $\mathrm{pwfl}(a) \in A$. The $\mathbb{Z}$-basis for $\mathcal{R}$ induced by the $\mathbb{Z}$-basis $\{X_1^{i_1} \cdots X_k^{i_k}\}$ for A, where $0 \le i_j \le \phi(M_j), 1 \le j \le k$, is referred to as the powerful basis for $\mathcal{R}$. The *powerful norm* is then defined as $|a|^{\mathrm{pwfl}} = |\mathrm{pwfl}(a)| \in \mathbb{Z}$.

The powerful norm is closely related to the canonical norm [13,15]. Specifically, it satisfies

$$|a|^{\mathrm{pwfl}} \le D_M |a|^{\mathrm{can}}, \tag{3}$$

where $D_M = \prod_{\text{prime } x|M} P(x)$ and $P(x) = \frac{2}{x \cdot \tan(\pi/2x)}$. Since $P(x) \le \frac{4}{\pi}$ for $x \ge 2$, the powerful norm is very close to the canonical norm when M has only a few prime factors [14].

In contrast, the infinity norm on the standard basis $[X^i]_{i=0,\ldots,\phi(M)-1}$ can be much larger than the canonical norm, depending on the number of prime factors in M (see [14] for details). Since the powerful norm is more tightly bounded than the standard norm, some noise-sensitive operations in BGV are performed on the powerful basis, such as decryption and bootstrapping.

2.3 BGV FHE Scheme

We introduce some fundamental operations used in the BGV scheme. The symbols involved are summarized in Table 2.

Table 2: Summary of symbols used in BGV.

Symbol	Meaning
M	Cyclotomic order of the polynomial ring
$\mathcal{R}$	Cyclotomic ring $\mathbb{Z}[X]/\Phi_M(X)$
p	Prime for plaintext modulus
r	Hensel lifting parameter for plaintext modulus
p^r	The plaintext modulus
d	The extension degree of each slot
n	The number of slots in R_{p^r}

Secret Key and Ciphertext. The secret key polynomial $s \in \mathcal{R}$ is sampled to have a bounded Hamming weight h. This means that h coefficients of s are set to -1 and 1 with equal probability, while the remaining coefficients are set to 0. For a plaintext polynomial $m \in \mathcal{R}_{p^r}$, we use

$$\mathrm{Enc}_s(m; p^r; \epsilon) = (b, a) = (-as + m + p^r\epsilon, a) \in \mathcal{R}_q^2 \tag{4}$$

to denote the BGV ciphertext that encrypts m, with noise $p^r\epsilon$, with respect to the secret key s. Here, q is coprime to p, a is sampled from $\mathcal{R}_q$, and ϵ is sampled from the discrete Gaussian distribution with standard deviation σ. Decryption of the ciphertext is performed by computing $[[\mathrm{pwfl}(b+as)]_q]_{p^r} = [m+p^r\epsilon]_{p^r} = m$. The notation can be simplified to $\mathrm{Enc}(m; p^r)$ or $\mathrm{Enc}(m)$ if the omitted parameters are clear from the context or unimportant.

The plaintext of BGV is often encrypted in a SIMD (Single Instruction, Multiple Data) manner. Specifically, using ring isomorphism, the plaintext space $\mathcal{R}_{p^r}$ can be represented as the direct product of $n = \frac{\phi(m)}{d}$ copies of finite fields (or finite rings if $r > 1$) with extension degree d, where d is equal to the multiplicative order of p in $\mathbb{Z}_m^*$. Thus, we may also use $\mathrm{Enc}([m_0, m_1, \ldots, m_{n-1}])$ to denote the encryption of the plaintext polynomial whose n slots hold the values $[m_0, m_1, \ldots, m_{n-1}]$.

Key Switching. For a given ciphertext $\mathrm{Enc}_s(m)$, it is feasible to switch it to $\mathrm{Enc}_{s'}(m)$ for another secret key s' using the key switching operation. This operation requires the encryption of the appropriate forms of s under the secret key s' in the public key.

Homomorphic Addition and Multiplication. Given $\mathrm{Enc}_s(m_0; p^r)$ and $\mathrm{Enc}_s(m_1; p^r)$, it is possible to compute homomorphically $\mathrm{Enc}_s(m_0 + m_1; p^r)$ and $\mathrm{Enc}_s(m_0 \cdot m_1; p^r)$. Additionally, homomorphic multiplication requires re-linearization, which is essentially key-switching from s^2 to s.

Noise Measurement. In BGV, the decryption result will be correct if $|m + p^r\epsilon|^{\mathrm{pwfl}} < q/2$. Furthermore, the ratio $q/|m + p^r\epsilon|^{\mathrm{can}}$, often referred to as the capacity of a ciphertext, is typically used to measure the capability of this ciphertext to perform homomorphic operations before a decryption error may occur.

BGV Bootstrapping. As homomorphic computations progress, the capacity of the ciphertext continues to decrease. Bootstrapping is required when the capacity becomes too low to perform any further homomorphic operations. Bootstrapping removes the noise in the input ciphertext by homomorphically evaluating the decryption function of the input ciphertext. Given $c = \mathrm{Enc}_{s'}(m; p^r)$ as the input ciphertext, BGV bootstrapping consists of the following five steps, as illustrated in Figure 1.

- *Simplifying the decryption formula:* By performing modulus-switching, key-switching and some scalar operations on c, we obtain $(b, a) \in \mathcal{R}^2$ satisfying $\mathrm{pwfl}(b + as) \equiv \mathrm{pwfl}(p^{e-r}m + \epsilon) \bmod p^e$ for some $e > r$. Let $m^* = b + as$, the decryption formula is simplified to perform digit removal on $\mathrm{pwfl}(m^*)$ to obtain $\mathrm{pwfl}(m) = \mathrm{pwfl}(m^*)\langle e-1, e-r\rangle_p$.
- *Homomorphic inner product:* Compute $b + a \cdot \mathrm{Enc}_{s'}(s; p^e)$. Since $\mathrm{Enc}_{s'}(s)$ has plenty of capacity, this step outputs $\mathrm{Enc}_{s'}(m^*; p^e)$ with high capacity.
- *Moving coefficients into slots:* A linear transformation called CoeffToSlot moves the coefficients of $\mathrm{pwfl}(m^*)$ into the slots. Then, the obtained ciphertext is unpacked into d ciphertexts, whose slots of the i-th ciphertext contain $[\mathrm{pwfl}(m^*)_{i \cdot n}, \ldots, \mathrm{pwfl}(m^*)_{i \cdot n + n - 1}] \in \mathbb{Z}_{p^e}^n$.
- *Homomorphic digit removal:* For each of the d ciphertexts obtained in the last step, the higher digits of their values in slots are extracted through a series of polynomial evaluations. This is also the most computationally heavy step. The i-th ciphertext now encrypts $[\mathrm{pwfl}(m)_{i \cdot n}, \ldots, \mathrm{pwfl}(m)_{i \cdot n + n - 1}] \in \mathbb{Z}_{p^r}^n$.
- *Moving slots back into coefficients:* Finally, the extracted digits in the slots of the d ciphertexts are packed into a single ciphertext. The slot values of the packed ciphertexts are moved into the powerful basis coefficients by another linear transform called SlotToCoeff.

Thin Bootstrapping. In 2018, Chen and Han discovered that when each slot only stores integers in $\mathbb{Z}_{p^r}$ instead of extension field (ring) elements, the procedures above can be simplified [6]. Specifically, the digit removal only needs to be applied once. Their modified bootstrapping routine consists of four steps. These steps are illustrated in Figure 2.

- *SlotToCoeff:* Move the integers in slots to the powerful basis coefficients, resulting in an encryption of a plaintext polynomial with sparse coefficients under the powerful basis.
- *Decryption Formula Simplification:* Simplify the decryption formula using the same technique of HS bootstrapping. Note that the modulus switching error produced in this step will make the encrypted plaintext polynomial non-sparse.
- *CoeffToSlot:* Move the powerful basis coefficients to the slots and project the values in slots to $\mathbb{Z}_{p^e}$ using a linear map.
- *Homomorphic Digit Removal:* Now, since each slot contains an integer instead of an extension field (ring) element, no unpacking is needed anymore. The digit removal is also applied on only one ciphertext instead of d ciphertexts.

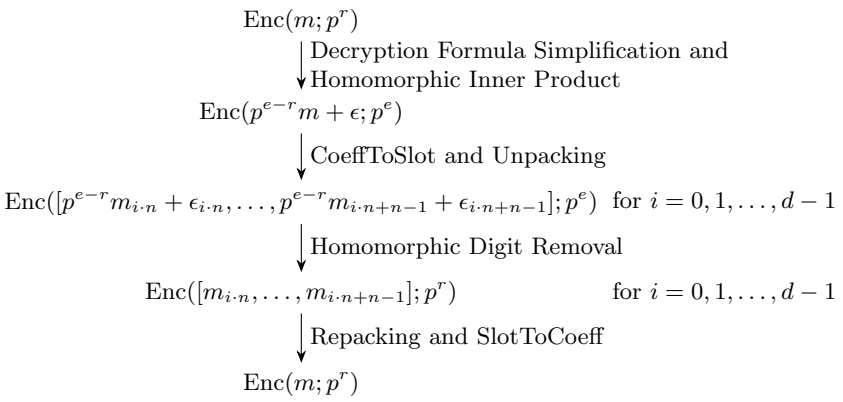


Fig. 1: The workflow of GV bootstrapping [9]. The workflow of HS bootstrapping [13,15] can be obtained by simply replacing e by $r + e - e'$. Note that both m and ϵ are represented in the powerful basis by default.

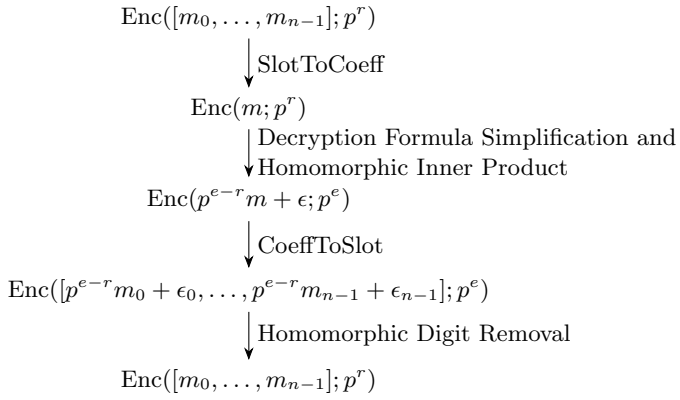


Fig. 2: The workflow of thin GV bootstrapping. The workflow of HS bootstrapping can be obtained by simply replacing e by $r + e - e'$. Both m and ϵ are represented in the powerful basis by default.

Overflow Part Extraction. In [17], Kim et al. propose an overflow part extraction technique to facilitate the bootstrapping of all RLWE-based FHE schemes with the functional bootstrap of FHEW/TFHE. When applied to a BGV ciphertext, the process takes as input a ciphertext $(b, a) = \mathrm{Enc}_s(m; p^r; \epsilon) \in \mathcal{R}_q^2$ satisfying $|m + p^r\epsilon| < \frac{q}{2\Delta}$ for some integer Δ. Define $(b', a') = ([\Delta b]_q, [\Delta a]_q) \in \mathcal{R}^2$ and $(b'', a'') = (\frac{b'-\Delta b}{q}, \frac{a'-\Delta a}{q})$. Then (b'', a'') extracts a noiseless encryption of the overflow part $I = \lfloor (b' + a's)/q \rceil$ modulo Δ. Specifically, by denoting $b + as = m + p^r\epsilon + Kq$, it has $b'' + a''s \equiv ((b' + a's) - \Delta(b + as))/q \equiv (Iq + \Delta(m + p^r\epsilon) - \Delta(m + p^r\epsilon + Kq))/q \equiv I \bmod \Delta$.

Moreover, by performing the above operations on the powerful basis instead of the standard basis, we can obtain a noiseless encryption of $I = \lfloor \mathrm{pwfl}(b' + a's)/q \rceil$. In this case, the input requirement should also be measured with respect to the powerful norm, i.e., $|m + p^r\epsilon|^{\mathrm{pwfl}} < \frac{q}{2\Delta}$.

2.4 Null Polynomial in $\mathbb{Z}_{p^e}[X]$

BGV bootstrapping involves evaluating polynomials in $\mathbb{Z}_{p^e}[X]$. When $e > 1$, $\mathbb{Z}_{p^e}$ is not a field, rendering the conventional polynomial theory over finite fields inapplicable to $\mathbb{Z}_{p^e}[X]$. One difference is that it is impossible to represent all functions over $\mathbb{Z}_{p^e}$ as polynomials in $\mathbb{Z}_{p^e}[X]$. A necessary but not sufficient condition exists for a function over $\mathbb{Z}_{p^e}[X]$ to be represented as a polynomial, as detailed in Lemma 1.

Lemma 1 ([8]). *Let $f \in \mathbb{Z}_{p^e}[X]$, then for all $a \in \mathbb{Z}$, it has*

$$f(a + p^k) \equiv f(a) \bmod p^k, \quad \forall k \leq e.$$

On the other hand, there exist low degree null polynomials in $\mathbb{Z}_{p^e}[X]$, which are non-zero polynomials but evaluate to zero at every point in $\mathbb{Z}_{p^e}$. The construction of null polynomials is closely related to p-adic valuation $v_p(\cdot) : \mathbb{Z} \to \mathbb{Z}$, which is defined as

$$v_p(x) = \begin{cases} \mathrm{argmax}_i(p^i | x) & \text{if } x \neq 0 \\ \infty & \text{if } x = 0 \end{cases}. \tag{5}$$

The most common way to construct null polynomials is to use the falling factorial basis $(X)_i$, which is defined as

$$(X)_i = \prod_{j=0}^{i-1} (X - i). \tag{6}$$

It can be verified that $(X)_i$ is a null polynomial over $\mathbb{Z}_{p^{v_p(i!)}}$ due to $i!|(a)_i$ for all $a \in \mathbb{Z}$.

The null polynomials can also be defined over $S \subseteq \mathbb{Z}_{p^e}$, which means they only need to evaluate to zero at every point in S instead of in $\mathbb{Z}_{p^e}$ [8]. Suppose there exists a monic null polynomial $f(X)$ over S with a small degree d, then we can represent any polynomial defined over S as a polynomial of degree at most $d - 1$ by reducing it modulo $f(X)$.

3 Homomorphic Digit Removal and its Optimizations

This section describes the existing digit removal algorithms in BGV [13,6,15,8]. Recall that, given an integer r and an encryption of $w \in \mathbb{Z}_{p^e}$, the homomorphic digit removal algorithm aims to output the encryption of the highest r digits $w\langle e-1, e-r\rangle_p$ by homomorphically computing and discarding the lowest $e-r$ digits one by one.

We start by introducing the homomorphic digit removal procedure used in HS bootstrapping. This procedure utilizes a lifting polynomial, denoted as $F_k(X)$, to perform digit removal. $F_k(X)$ is a degree-p polynomial over $\mathbb{Z}$ satisfing the property $F_k(a + bp^j) \equiv a \bmod p^{j+1}$ for all $a \in [\![p]\!], b \in \mathbb{Z}$ and $1 \leq j \leq k$, where k is a positive integer such that $k \leq e-1$. Let $w_{i,j}$ be an integer modulo p^{e-i} whose lowest digit is $w\langle i\rangle_p$ and the higher j digits are zero. It follows that

$$w_{i,j} = F_k(w_{i,j-1}), \forall 1 \leq j \leq k. \tag{7}$$

Furthermore, HS bootstrapping uses the formula

$$w_{i,0} = (w - \sum_{j=0}^{i-1} w_{j,i-j} \cdot p^j)/p^i \tag{8}$$

to shift the i-th digit down to the lowest digit. These two equations are sufficient to compute all $w_{i,e-1-i}$ for $i < e-r$. Finally, the highest r digits are extracted by removing these $e-r$ lowest digits:

$$w\langle e-1, e-r\rangle = (w - \sum_{i=0}^{e-r-1} w_{i,e-1-i} \cdot p^i)/p^{e-r}. \tag{9}$$

An example for $e = 5, r = 2$ is presented in Figure 3. The horizontal arrows represent evaluations of F_k, while the diagonal arrows indicate the computation of $w_{i,0}$ and $w\langle e-1, e-r\rangle$ from previous rows using Equation (8) and Equation (9).

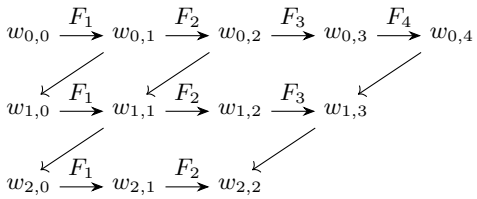


Fig. 3: Digit removal in HS bootstrapping for $e = 5, r = 2$.

Chen-Han Optimization. Chen and Han [6] discovered the existence of a digit extraction polynomial $G_e(X)$ of degree $(p-1)(e-1)+1$, satisfying $G_e(a) \equiv [a]_p \bmod p^e$ for all $a \in \mathbb{Z}$. This discovery allows for the computation of $w_{i,j}$ to be skipped when $e-i-1-r < j < e-i-1$, and $w_{i,e-1-i} = G_{e-i}(w_{i,0})$ can be computed directly. As a result, the multiplicative depth required to extract $w_{i,e-1-i}$ is reduced from $(e-1-i)\log_2(p)$ to $\log_2((p-1)(e-i-1)+1)$. This reduction increases the capacity of the ciphertext after bootstrapping.

An illustration of the Chen-Han optimization for $e=5, r=2$ is presented in Figure 4. In this scenario, the computations for $w_{0,3}, w_{1,2}, w_{2,1}$ are skipped, and $w_{0,4}, w_{1,3}, w_{2,2}$ are directly computed using G_5, G_4, G_3.

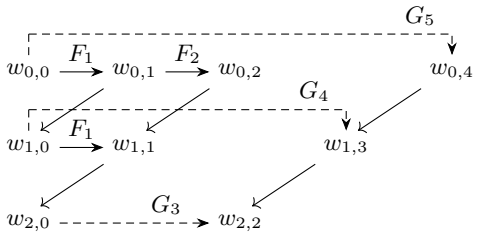


Fig. 4: Chen-Han digit removal for $e=5, r=2$.

GIKV Optimizations. Geelen et al. [8] applied several optimizations to the digit removal algorithm:

- They noted that for a fixed i, $w_{i,j} = G_{i+j}(w_{i,0})$ are computed as evaluations of multiple polynomials on the same input $w_{i,0}$. The multi-polynomial Baby-Step-Giant-Step (BSGS) algorithm can be used to reduce the complexity of evaluating m degree-n polynomials to $2\sqrt{mn}$ [11]. Furthermore, if the evaluated polynomials are odd functions, which is the case for $G_e(X)$, the complexity can be reduced to $\sqrt{2mn}$.
- They observed that $w_{i,j}$ satisfies the properties of $w_{i,j'}$ for $j' \leq j$. Thus, it suffices to compute only $w_{i,j}$ instead of computing all $w_{i,j'}$ for $j' < j$. This optimization must be applied carefully to ensure that the digit removal procedure's overall depth (or the critical path's length) is not increased.
- They decomposed $G_e(X)$ into the composition of multiple polynomials, reducing the number of non-scalar multiplications needed to evaluate $G_e(X)$ from $O(\sqrt{pe})$ to $O(\sqrt{p}\sqrt[4]{e})$, at the cost of increasing the depth by $\log_2(p)$.
- They represented the null polynomials as a lattice and found a small-norm representation of $G_e(X)$ and $F_k(X)$ by solving the CVP problem in that lattice. This helps reduce the capacity consumed in homomorphic polynomial evaluation due to smaller polynomial coefficients.

4 Details on Our Digit Removal Procedure

In this section, we elucidate the specifics of our optimization of the digit removal procedure. Initially, we offer a summary of BGV bootstrapping equipped with our optimized digit removal procedure in Section 4.1. Then we present the construction of low-degree null polynomials in Section 4.2 and provide the determination of bootstrapping parameters in Section 4.3. Finally, we explain how to adapt our method for thin bootstrapping in Section 4.4, and discuss how our method can be combined with other existing optimizations in Section 4.5.

4.1 Overview of BGV Bootstrapping with Our Digit Removal

To begin with, we describe our bootstrapping procedure in detail, since it modifies the original bootstrapping procedure and involves some extra parameters. We assume that the ciphertext to be bootstrapped encrypts extension field (ring) elements in its slots, and we leave the thin bootstrapping case for Section 4.4. The workflow of our bootstrapping is described below and illustrated in Figure 5.

Decryption Formula Simplification. Let $\mathrm{Enc}_{s'}(m; p^r)$ denote the ciphertext that is to be bootstrapped. We first modulus-switch it to q_{ks}, and obtain $\mathrm{Enc}_{s'}(m; p^r)$ modulo q_{ks}. Next, we apply key-switching to this ciphertext to obtain a new ciphertext with respect to the bootstrapping key s, denoted as $\mathrm{Enc}_{s}(m; p^r)$ modulo $q_{ks}R$, where R represents an additional key-switching modulus. Then we perform modulus-switching on the powerful basis to q_0, yielding a ciphertext $(b, a) \in \mathcal{R}_{q_0}^2$. For a more concise statement of our bootstrapping procedure, the selection of parameters q_{ks}, q_0 and R, along with the specific key-switching procedure, will be detailed in Section 4.3.

Suppose Δ is an auxiliary modulus. The ciphertext $(b, a) \in \mathcal{R}_{q_0}^2$ is multiplied by Δ to obtain $(b', a') = ([\Delta b]_{q_0}, [\Delta a]_{q_0}) \in \mathcal{R}^2$, where both $\mathrm{pwfl}(b')$ and $\mathrm{pwfl}(a')$ are q_0-reduced. The modulus of (b', a') is then directly raised from q_0 to a very large modulus Q. Next, the ciphertext is modulus-switched and key-switched to the main key s', resulting in a ciphertext $\tilde{c} = \mathrm{Enc}_{s'}(\Delta m + [q_0]_{p^{r+v_p(\Delta)}} I'; p^{r+v_p(\Delta)})$, where $I' = \lfloor \mathrm{pwfl}(b' + a's)/q_0 \rceil$ is the overflow part of (b', a') on the powerful basis with respect to q_0.

On the other hand, utilizing the overflow part extraction technique in [17] (as described in Section 2), a noiseless encryption of I' with respect to s is obtained by computing $(b'', a'') = (\frac{b' - \Delta b}{q_0}, \frac{a' - \Delta a}{q_0}) \in \mathcal{R}_{\Delta}^2$, where both $\mathrm{pwfl}(b'')$ and $\mathrm{pwfl}(a'')$ are required to be Δ-reduced.

Homomorphic Inner Product. Compute $c'' = b'' + a'' \cdot \mathrm{Enc}_{s'}(s; p^{r+v_p(\Delta)}) = \mathrm{Enc}_{s'}(I' + \Delta I^*; p^{r+v_p(\Delta)})$, where $I^* = \lfloor \mathrm{pwfl}(b'' + a''s)/\Delta \rceil$ denotes the overflow part of (b'', a'') on the powerful basis with respect to Δ.

CoeffToSlot and Unpacking. Move the coefficients of $\mathrm{pwfl}(I' + \Delta I^*)$ into the slots by applying a linear transform to c''. Then, unpack the resulting ciphertext into d ciphertexts, each one encrypting only integers in its slots.

Digit Removal. Each value stored in the slots currently is equal to one of the coefficients of $\mathrm{pwfl}(I' + \Delta I^*)$ and has the form of $J = J' + \Delta J^*$ for some integer J', J^*. We note that the plaintext modulus here is $p^{r+v_p(\Delta)}$, and the goal of the digit removal procedure is to extract J' modulo $p^{r+v_p(\Delta)}$ from $[J' + \Delta J^*]_{p^{r+v_p(\Delta)}}$. We assume that J' and J^* are bounded by B, the derivation of which will be discussed in Section 4.3. Our digit removal procedure depends on the selection of Δ, which has two cases.

In the first case, we choose an integer t such that $p^t > 2B+1$, and let $\Delta = p^t$. The digit removal involves computing $J\langle t-1, 0\rangle$ modulo p^{r+t} from J for all the slot values. This can be accomplished by using the digit removal algorithms in [13,6], but with acceleration by the low-degree null polynomials constructed in Section 4.2.

In the second case, we require the existence of an integer Δ_0 such that Δ_0 is coprime with p and $(p-1)/(2B) - 1 > \Delta_0 > (2B+1)$. We then let $\Delta = \Delta_0$. The digit removal involves computing J' from $J' + \Delta_0 J^*$, which is accomplished by evaluating the lifted interpolation polynomial constructed in Section 4.2.

Note that the computation above is applied to all the d ciphertexts after unpacking.

Repacking and SlotToCoeff. Pack the d ciphertexts into a single one and move the encrypted slot values into the powerful basis coefficients through a linear transform. The resulting ciphertext is $c_I = \mathrm{Enc}_{s'}(I'; p^{r+v_p(\Delta)})$.

Assembling the Parts Together. The process varies depending on the choice of Δ. If $\Delta = p^t$, we compute $(\tilde{c} - [q_0]_{p^{r+t}} c_I)/p^t$, which yields $(\mathrm{Enc}_{s'}(p^t m + [q_0]_{p^{r+t}} I'; p^{r+t}) - [q_0]_{p^{r+t}} \mathrm{Enc}_{s'}(I'; p^{r+t})))/p^t = \mathrm{Enc}_{s'}(p^t m; p^{r+t})/p^t = \mathrm{Enc}_{s'}(m; p^r)$.

On the other hand, if $\Delta = \Delta_0$, we compute $(\tilde{c} - [q_0]_{p^r} c_I) \cdot [\Delta_0]_{p^r}^{-1}$, which results in $(\mathrm{Enc}_{s'}(\Delta_0 m + [q_0]_{p^r} I'; p^r) - [q_0]_{p^r} \mathrm{Enc}_{s'}(I'; p^r)) \cdot [\Delta_0]_{p^r}^{-1} = \mathrm{Enc}_{s'}(\Delta_0 m; p^r) \cdot [\Delta_0]_{p^r}^{-1} = \mathrm{Enc}_{s'}(m; p^r)$.

4.2 Constructing Low-degree Polynomials for Digit Removal

Denote $t = v_p(\Delta)$ and $\Delta_0 = \Delta/p^t$. It should be noted that the definition of Δ_0 aligns with that in Section 4.1, and Δ_0 is coprime to p. Recall that the goal of the digit removal procedure is to extract J' modulo p^{r+t} from $[J' + \Delta J^*]_{p^{r+t}}$ provided that $|J'| < \Delta/2$. As discussed in Section 4.1, the approach to accomplish this differs depending on the value of t. For $t > 0$, (i.e., the $\Delta = p^t$ case), we can construct two types of low-degree null polynomials by exploiting the sparsity of the input to digit removal, namely the *global* ones from the sparsity of the whole input, and the *local* ones from the sparsity of the lower digits in the input. For $t = 0$, (i.e., the $\Delta = \Delta_0$ case), although existing digit removal polynomials are unavailable, we can still construct a digit extraction polynomial through interpolation. This case can also be viewed as a special case of the global null polynomial.

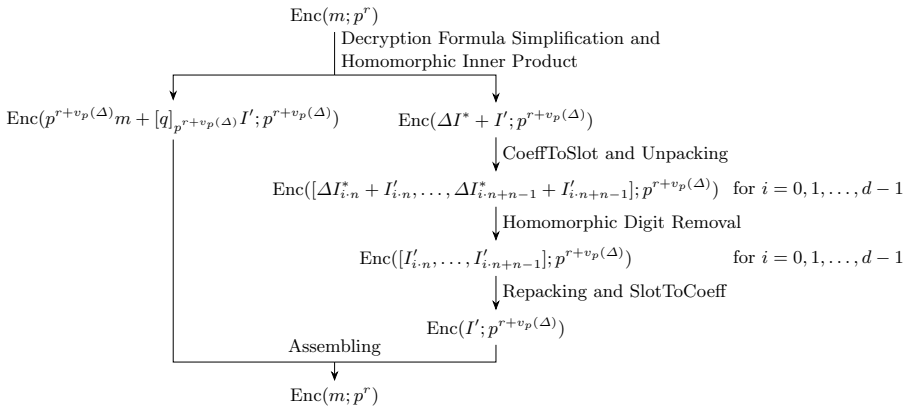


Fig. 5: The workflow of BGV bootstrapping using our digit removal method. m, ϵ, I', I^* are all represented in the powerful basis.

Global Null Polynomials from Sparsity of the Input. We begin with the case where $t > 0$. In this case, computing J' is equivalent to extracting the lowest t digits from $[J]_{p^{r+t}}$. This can be accomplished by evaluating the digit extraction or lifting polynomials, as introduced in [13,6] and described in Section 3, provided that $J' = [J']_{p^t}$. Subsequently, we demonstrate how to construct global null polynomials to reduce the degree of these polynomials.

We adopt the notations $w_{i,j}$ from Section 3 and let $w_{0,0} = w = J' + \Delta J^*$. Since $|J'| \leq B$ and $|J^*| \leq B$, it has that $\mathrm{supp}(w) = \mathrm{supp}(J' + \Delta J^*) \leq (2B+1)^2$. It is evident that all $w_{i,j}$'s are deterministic functions of w because they are obtained by evaluating polynomials with predetermined coefficients and division by power of p. Therefore, $\mathrm{supp}(w_{i,j}) \leq \mathrm{supp}(w) \leq (2B+1)^2$ holds for all $w_{i,j}$. When a polynomial $f(X)$ is evaluated on $w_{i,j}$ (where $f(X)$ could be either a digit extraction or lifting polynomial), $g(X) = \prod_{a \in \mathrm{supp}(w_{i,j})}(X - a)$ serves as the global monic null polynomial over $\mathrm{supp}(w_{i,j})$ with a degree at most $(2B+1)^2$. By reducing $f(X)$ modulo $g(X)$, we obtain an equivalent representation of $f(X)$ with a degree less than $(2B+1)^2$. This concludes the following lemma.

Lemma 2. *For $t > 0$, it has $|supp(w_{i,j})| \leq (2B+1)^2$ during the digit removal procedure of BGV bootstrapping. Moreover, all polynomials evaluated on $w_{i,j}$ can be reduced to a polynomial with a degree less than $(2B+1)^2$.*

In the alternative case where $t = 0$, J' is not equal to the lower digits of J since Δ_0 is coprime with p. This implies that the computation of J' using existing digit removal algorithms is not feasible. Nevertheless, we demonstrate that it is still possible to construct a low-degree polynomial by leveraging the sparsity of the input.

Lemma 3. *For $t = 0$, there exists a polynomial f of degree less than $(2|J'| + 1)(2|J^*| + 1) \leq (2B+1)^2$, such that $f(J' + \Delta_0 J^*) = J'$ over $supp(J' + \Delta_0 J^*)$, provided that $|J' + \Delta_0 J^*| < p/2$ and $|J'| < \Delta_0/2$.*

Proof. $|J'| < \Delta_0/2$ guarantees the information of J' is recoverable from $J' + \Delta_0 J^*$. Since $|J' + \Delta_0 J^*| < p/2$ and $\mathbb{Z}_p$ is a field, the standard interpolation theory guarantees the existence of $f_0 \in \mathbb{Z}_p[X]$ such that $f_0(J' + \Delta_0 J^*) = J'$.

We can lift this interpolation to $\mathbb{Z}_{p^r}[X]$ with $r > 1$ recursively. Given a polynomial $f_k \in \mathbb{Z}_{p^k}[X]$ over the set $S = \mathrm{supp}(J' + \Delta_0 J^*) \subseteq [\![p]\!]$, we aim to lift it to $f_{k+1} \in \mathbb{Z}_{p^{k+1}}[X]$ such that $f_{k+1}(a) \equiv [f_k(a)]_{p^k} \bmod p^{k+1}$. Let $d(a) = [(f_k(a) - [f_k(a)]_{p^k})/p^k]_p$, there exists a degree $|S| - 1$ polynomial $f_k^* \in \mathbb{Z}_p[X]$ such that $f_k^*(a) \equiv d(a)$ for all $a \in S$, which can be computed using interpolation algorithms over $\mathbb{Z}_p$. Let $f_{k+1} = f_k - p^k \cdot f_k^*$. It can be verified that $f_{k+1}(a) \equiv [f_k(a)]_{p^k} \bmod p^{k+1}$ for all $a \in S$, which finishes the induction. □

We note that the condition $|J' + \Delta_0 J^*| < p/2$ appears to be essential in our construction. Specifically, suppose $|J' + \Delta_0 J^*| \geq p/2$ and we attempt to construct an f as in Lemma 3. If $p > \Delta_0$, there may exist $J_0, J_1 \in \mathrm{supp}(J' + \Delta_0 J^*)$ such that $J_0 = J_1 + kp$ for some $k \neq 0$ coprime to Δ_0. Consequently, we have $f(J_0) \equiv f(J_1) \bmod p$ by Lemma 1, which implies that $f(J_0) = f(J_1)$ since $f(J_0), f(J_1) \in [\![\Delta_0]\!] \subset [\![p]\!]$. However, from the definition of f, we have $f(J_0) - f(J_1) \equiv J_0 - J_1 = kp \bmod \Delta_0$, leading to a contradiction. For $p < \Delta_0$, we can consider $J_0 = J_1 + k\Delta_0$ with $k \neq 0$ coprime to p and obtain a contradiction similarly.

It appears that the interpolated polynomial $f(X)$ for $t = 0$ is irrelevant to the global null polynomial. In fact, the construction of $f(X)$ has already accounted for the reduction of the corresponding global null polynomial $g(X) = \prod_{a \in \mathrm{supp}(J' + \Delta_0 J^*)}(X - a)$, which also has a degree of at most $(2B + 1)^2$. Consequently, we categorize the method for the case of $t = 0$ as a part of the global null polynomial optimization.

Local Null Polynomial From Small Lower Digits. The construction of our local null polynomials is analogous to the method employed by Geelen et al. [8]. When applied to our case for reducing the degree of the digit removal polynomials, the local null polynomials possess a lower degree than the global null polynomials for small parameters but exhibit a higher degree asymptotically. Note that the local null polynomials are only applicable in the case where $\Delta = p^t$. The detailed construction is as follows.

Recall that in our digit removal process, the value of the lower digits to be removed has a support size of $2B + 1$, which is independent of the message m and the noise ϵ of the input ciphertext. This is a significant difference from previous digit removal methods, where the support size of the lower digits to be removed is dependent on both m and ϵ. For instance, the powerful norm of lower digits in HS bootstrapping is $B_0(1 + \delta + p^{-e'}(2p^r + 1 + 1/q^2))$ [15], while in GV bootstrapping it is $B_0 + |p^{e-r}m + p^e\epsilon|/q$ [9]. This means that the support size of the lower digits in our bootstrapping can be much smaller than that in previous approaches. This property can be leveraged to construct low-degree null polynomials as stated in the following lemma.

Lemma 4. *Suppose p is a prime and $w \in \mathbb{Z}_{p^e}$. If $L \le w\langle t-1, 0\rangle_p \le H$,*

$$\Lambda_k(X) = \prod_{j=0}^{k-1}(\prod_{i=L}^{H}(X-i) - j \cdot p^{t+v_p((H-L)!)-\lfloor \log_p(H-L)\rfloor})$$

is a monic null polynomial over $supp(w) \subseteq \mathbb{Z}_{p^e}$ for $e \le k(t + v_p((H-L)!) - \lfloor \log_p(H-L)\rfloor) + v_p(k!)$.

Proof. Denote $g(X) = \prod_{i=L}^{H}(X-i)$ and $w_t = w\langle t-1, 0\rangle$. Then it has

$$\begin{aligned} v_p(g(w)) &= v_p(w-w_t) + v_p(\prod_{i=L}^{w_t-1}(w-i)) + v_p(\prod_{i=w_t+1}^{H}(w-i)) \\ &= v_p(w-w_t) + v_p(\prod_{j=w-w_t+1}^{w-L} j) + v_p(\prod_{j=w-H}^{w-w_t-1} j). \end{aligned}$$

Since $H - L \le p^t - 1$, $p^t \nmid j = w - i$ except for $j = w - w_t$ (i.e., $i = w_t$). We also know that $v_p(a) = v_p(a + b \cdot p^t)$ if $p^t \nmid a$, where $a, b \in \mathbb{Z}$. This implies $v_p(\prod_{j=w-w_t+1}^{w-L} j) + v_p(\prod_{j=w-H}^{w-w_t-1} j) = v_p(\prod_{j=1}^{w_t-L} j) + v_p(\prod_{j=w_t-H}^{-1} j)$ because

$$\begin{aligned} v_p(\prod_{j=w-w_t+1}^{w-L} j) &= v_p(\prod_{j=w-w_t+1}^{w-L}(j+w_t-w)) = v_p(\prod_{j=1}^{w_t-L} j), \\ v_p(\prod_{j=w-H}^{w-w_t-1} j) &= v_p(\prod_{j=w-H}^{w-w_t-1}(j+w_t-w)) = v_p(\prod_{j=w_t-H}^{-1} j). \end{aligned}$$

Denote $h = w_t - L$ and $l = -(w_t - H)$, it only remains to derive a lower bound on $v_p(h!) + v_p(l!)$ where $h, l \in [0, H-L]$ and $h + l = H - L$. To do this, we generalize the definition of (unsigned) p-ary integers by allowing each digit to lie in $[0, 2p-2]$ instead of $[0, p-1]$. The digit symbol $\cdot\langle\cdot\rangle$ is generalized as well, and $a = \sum_i p^i a\langle i\rangle$ still holds for any a. We also define a function f_v on a generalized p-ary integer a as $f_v(a) = \sum_{i=1}^{k} a\langle k, i\rangle$, where a has $k+1$ digits. Obviously, $f_v(a) = v_p(a!)$ for a regular p-ary integer a.

Suppose the p-ary representation of both h and l has $k+1$ digits, we construct a generalized unsigned p-ary integer a with $k+1$ digits, where $a\langle i\rangle = h\langle i\rangle + l\langle i\rangle \in [0, 2p-2]$. Since $a\langle k, i\rangle = h\langle k, i\rangle + l\langle k, i\rangle$, it has $f_v(a) = f_v(h) + f_v(l)$.

Now, we normalize a into a regular p-ary integer following the textbook add-with-carry approach. Specifically, we find the lowest digit that is at least p, carry this digit to the next digit, and repeat this process until all digits lie in $[0, p-1]$. Suppose $a\langle i\rangle \ge p$, after a single step of carrying a becomes a', where a' represents the same integer as a and agrees with a in all but two digits, namely $a'\langle i\rangle = a\langle i\rangle - p$ and $a'\langle i+1\rangle = a\langle i+1\rangle + 1$. Now $f_v(a') - f_v(a) = a'\langle k, i+1\rangle - a\langle k, i+1\rangle = 1$. It takes at most $\lfloor \log_p(H-L)\rfloor$ carries to normalize a into a regular p-ary representation of $h + l = H - L$, implying $f_v(H-L) \le \lfloor \log_p(H-L)\rfloor + f_v(h) + f_v(l)$.

The above argument establishes that $v_p(g(w)) = v_p(w-w_t)+v_p(h!)+v_p(l!) \geq t+v_p((H-L)!)-\lfloor \log_p(H-L) \rfloor$. Suppose $v_p(g(w)) = a \cdot p^{t+v_p((H-L)!)-\lfloor \log_p(H-L) \rfloor}$, then $v_p(\Lambda_k(w)) = v_p(\prod_{j=0}^{k-1}(a-j)) + k(t+v_p((H-L)!) - \lfloor \log_p(H-L) \rfloor) \geq v_p(k!) + k(t+v_p((H-L)!) - \lfloor \log_p(H-L) \rfloor)$. Thus, $\Lambda_k(w) \equiv 0 \bmod p^e$ for $e \leq v_p(k!) + k(t+v_p((H-L)!) - \lfloor \log_p(H-L) \rfloor)$, which is equivalent to saying that $\Lambda_k(X)$ is a null polynomial over $\mathrm{supp}(w)$ modulo p^e. □

Remark. Lemma 4 can be viewed as an extension of the construction of null polynomials presented in [8]. Specifically, in [8] the lowest digit can take any value while the remaining $t-1$ lowest digits are required to be 0, which corresponds to $H = -L = (p-1)/2$ for odd p, or $H = 1, L = 0$ for $p = 2$ in Lemma 4.

Next, we demonstrate how Lemma 4 can be utilized to expedite the digit removal methods in Section 3. To begin with, we depict how local null polynomials can be employed to accelerate the evaluation of the digit removal polynomials with input $w_{i,0}$, the first element in each row of the digit removal methods.

For the first row of the digit removal in Figure 3 (HS digit removal), Figure 4 (Chen-Han digit removal), or GIKV's first optimization on digit removal, Lemma 4 can be invoked to speed up the polynomial evaluations on $w_{0,0}$. In this case, $\Lambda_{\lceil e/t \rceil}(X)$ with $H = -L = B$ is a monic null polynomial of degree $\lceil \frac{e}{t} \rceil (2B+1)$ over $\mathrm{supp}(w_{0,0})$ because $|w_{0,0}\langle t-1, 0\rangle_p| = |I'| = B$. Polynomial evaluations on $w_{0,0}$ can be accelerated if this null polynomial has a lower degree than the lifting polynomial $F(X)$ or the digit extraction polynomial $G_i(X)$. Selecting a larger t will diminish both the computational complexity and the depth consumption of the first-row digit removal because the degree of $H(X)$ is reduced. However, it also augments the plaintext modulus $p^e = p^{r+t}$ for the SlotToCoeff transform and the digit removal steps, which may result in less after-bootstrap capacity.

For the i-th row of digit removal ($i > 0$), the local null polynomial cannot be directly constructed for $w_{i,0}$ because $w_{i,0}\langle t-1, 0\rangle$ for $i > 0$ are no longer bounded by $|B|$ and its support size may not be small. To tackle this issue, we can modify the digit shifting formula Equation (8) into the following formula Equation (10). Note that $w_{i,0}$ obtained in this way can still be used in digit removal and additionally satisfy $|w_{i,0}\langle t_i - 1, 0\rangle| = |w\langle t_i - 1 + i, i\rangle|$ for some $t_i \leq t - i$.

$$w_{i,0} = (w - \sum_{j=0}^{i-1} w_{j,i-j+t_i-1} \cdot p^j)/p^i. \tag{10}$$

Specifically, for $t_i = t-i$, $|w_{i,0}\langle t-1, 0\rangle| \leq \lceil \frac{B}{p^i} \rceil$. Although this approach reduces the cost of evaluating polynomials on $w_{i,0}$, the depth and computational cost of obtaining $w_{i,0}$ may be increased.

Finally, for polynomials evaluated on the other $w_{i,j}$'s with $j > 0$ in HS or Chen-Han digit removal, we can adopt $\Lambda_{\lceil e/(j+1) \rceil}(X)$ with $H = \min(\lfloor p/2 \rfloor, B)$, $L = \max(-\lfloor (p-1)/2 \rfloor, -B)$ as the local null polynomial over $\mathrm{supp}(w_{0,j})$, thereby accelerating the polynomial evaluation.

Remark. Unlike the global null polynomials, the construction of local null polynomials is independent of the overflow part extraction method of [17]. We can obtain the same bound B on the lower digits of the input of the digit removal step in HS/GV bootstrapping by modifying their parameter selection criteria, without using the techniques in [17]. This enables us to construct low-degree local null polynomials similarly. Please refer to Supplementary Material B for more details.

4.3 Determining the Parameters

We proceed to show how to determine the parameters involved in our bootstrapping procedure, as detailed in Section 4.1. Initially, we offer a concise introduction to the key-switching procedure and give a discussion regarding the choice of corresponding parameters.

Details on Key-switching. The key-switching algorithm is the same as that used in HElib [14]. Let $\mathrm{Enc}_{s'}(m; p^r; \epsilon) = (b, a) \in \mathcal{R}_{q_{ks}}^2$ be the input ciphertext before key-switching to s. Similar to HElib, we assume ϵ has the size of a modulus-switching noise. q_{ks} is decomposed in a mixed-radix way during key-switching. Specifically, let $D_i (1 \leq i \leq L)$ be the radices where D_1 is the lowest radix and $\prod_{i=1}^{L} D_i = q_{ks}$. Also define $D_i^* = \prod_{j=1}^{i-1} D_j$. These is a unique decomposition of any $a \in \mathcal{R}_{q_{ks}}$ under this set of radices, i.e.,

$$a = \sum_{i=1}^{L} a_i D_i^*, a_i \in [\![D_i]\!]. \tag{11}$$

The key-switching keys from s' to s consist of $\mathrm{Enc}_s(D_i^* R s'; p^r; \epsilon_i)$ under modulus $q_{ks}R$, where R is the additional key-switching modulus coprime to p and ϵ_i's are fresh encryption noises.

The key-switching step computes

$$\mathrm{Enc}_{s'}(Rm; p^r; \epsilon') = (bR, 0) + \sum_{i=1}^{L} a_i \mathrm{Enc}_s(D_i^* R s'; p^r; \epsilon_i). \tag{12}$$

The noise ϵ' after key-switching consists of two parts, the scaled noise $R\epsilon$ and the key-switching-added noise $\sum_{i=1}^{L} a_i \epsilon_i$. Let β and α be the *canonical* bounds on $p^r \epsilon$ and $p^r \sum_{i=1}^{L} a_i \epsilon_i$, respectively. Then the additional key-switching modulus R is chosen such that the key-switching-added noise is smaller than the scaled noise, i.e., $R\beta \geq \alpha$, ensuring at most one bit of capacity is lost during key switching. This choice of R based on one-bit capacity loss is similar to that in HElib. Halevi and Shoup have established estimates of α and β [14], which are presented as follows. In our parameter selection, we set the k in these estimates to 10, as that in [14].

Lemma 5 ([14]). *With failure probability no more than* $erfc(k/\sqrt{2}) \cdot \phi(M)/2$, *the estimates of* β *and* α *are*

$$\beta = kp^r \sqrt{\frac{h\phi(M)\log(\phi(M))}{12}},$$

$$\alpha \leq \begin{cases} \max(D_i) Lp^r \sigma k \phi(M) \sqrt{\log(\phi(M))/12} & \text{if } M \text{ is a power of 2} \\ \max(D_i) Lp^r \sigma k M \sqrt{\phi(M)\log(\phi(M))/12} & \text{otherwise} \end{cases},$$

where $\sigma = 3.2$ *is the basic standard deviation of* ϵ_i*'s.*

Establish the Powerful Bound B. As mentioned earlier, the powerful bound B of I' and I^* determines the degree of the global null polynomial, which further affects the complexity of the digit removal procedure. Note that both I' and I^* can be represented as $\lfloor \text{pwfl}(b/q + as/q) \rceil$, where $\text{pwfl}(b), \text{pwfl}(a)$ can be viewed as random q-reduced polynomials in the powerful basis for some q and s is the secret key. Halevi and Shoup provided a high-probability bound on $\text{pwfl}(b/q + as/q)$, which we present as follows.

Lemma 6 ([15]). *Let* a *and* b *be random elements in* $\mathcal{R}$*, whose powerful basis coefficients are sampled uniformly in* $[\![q]\!]$*. Let* s *be a secret key with Hamming weight* h*, then*

$$\Pr[|b/q + as/q|^{pwfl} > (1 + 1/q^2)(k\sqrt{\frac{h\phi(M)2^{\omega(M)}}{12M}} + 0.5)] \lessapprox \phi(M) \cdot erfc(k/\sqrt{2}),$$

where $\omega(M)$ *is the number of distinct prime factors of* M*.*

Since the correction factor $1 + 1/q^2$ is very close to 1 in practice, we omit it and denote the bound on $\text{pwfl}(b/q + as/q)$ as B_0, and define $B = \lfloor B_0 \rceil$, i.e.,

$$B_0 = k\sqrt{\frac{h\phi(M)2^{\omega(M)}}{12M}} + 0.5 \text{ and } B = \lfloor B_0 \rceil.$$

In our parameter selection, M is chosen such that it has only 2 prime factors, ensuring that $\phi(M)2^{\omega(M)}/(12M)$ is bounded by $1/3$. On the other hand, k is a constant that is typically set to around 10. As a result, B can be considered to be proportional to $\sqrt{h}$.

Note that HElib sets the parameter k in Lemma 6 to 10 by default [16], corresponding to a failure probability $\approx 2^{-60}$ for $\phi(M) \leq 2^{16}$. In contrast, we use $k = 8$ to establish B in our analysis for better efficiency. Although this may increase the failure probability to a maximum of 2^{-33}, we argue that this is still practical as the failure probability remains smaller than those in most works on CKKS bootstrapping.

Employ the Sparse Secret Key Encapsulation. Since B increases with h and a smaller B leads to smaller degrees of null polynomials as described in Section 4.2, we adopt the sparse secret key encapsulation technique of Bossuat et al. [4] to key-switch to a bootstrapping-only secret key with a very low Hamming weight before simplifying the decryption formula. This will drastically reduce the value of h and thus the value of B.

The sparse key encapsulation technique requires key-switching keys from the main secret key s' (with Hamming weight h') into the bootstrapping secret key s (with Hamming weight h). To provide enough security, these key-switching keys should have a very low ciphertext modulus, which we denote as q_{boot}. Denoting the maximum ciphertext modulus for s' as Q', the security of the whole scheme is the minimum security in these two parameter sets: (M, h', Q') and (M, h, q_{boot}). While a small q_{boot} may provide higher security, it also needs to be large enough so that the noise during decryption formula simplification will not corrupt the correctness of the ciphertext.

Denote the modulus under which the decryption formula simplification takes place as q_0 and the modulus for key-switching as q_{ks}. The ciphertext to be bootstrapped is first modulus-switched to q_{ks}, key-switched to s (with a ciphertext modulus of $q_{boot} = q_{ks}R$ now), and finally modulus-switched to q_0 before the decryption formula simplification step. Thus, we need to take into account the modulus-switching error and key-switching error when choosing q_{boot}.

Once q_{ks} is known, we can decide the value of $R \geq \alpha/\beta$ because α and β are determined by q_{ks}. For simplicity, we set $q_0 = q_{ks}$. Denote the ciphertext before overflow part extraction as $\mathrm{Enc}_s(m; p^r; \epsilon)$ under modulus q_0. We note that the condition $|\Delta(m + p^r\epsilon)|^{\mathrm{pwfl}} < q_0/2$ must be fulfilled so that the method of Kim et al. [17] works correctly, where Δ is the auxiliary modulus for overflow part extraction. The condition that q_0 needs to satisfy is presented in Lemma 7.

Lemma 7. *The overflow part extraction (in Section 2) works correctly during decryption formula simplification if*

$$q_0 > 2\Delta(2\beta D_M + B_0 p^r),$$

where D_M is the multiplicative factor when converting from the canonical norm to the powerful norm, as defined in Section 2.2.

Proof. As mentioned above, the input ciphertext of the overflow part extraction step must have its noise bounded by $\frac{q_0}{2\Delta}$ on the powerful basis. Thus, we will compute an estimate of this noise from the beginning.

The input ciphertext of bootstrapping has a modulus of q_0 and a noise with canonical norm β. After key-switching to the bootstrapping secret key s, the ciphertext has a modulus of q_0R and a noise bounded by $2\beta R$ on the canonical embedding, which is ensured by our choice of R. Then, we modulus-switch the ciphertext to q_0 on the powerful basis. The noise added by this modulus switching can be modeled as $\epsilon_0 + \epsilon_1 s$, where ϵ_i are powerful basis polynomials with coefficients uniformly distributed in $[-p^r/2, p^r/2]$. Thus, Lemma 6 can be applied to bound the modulus-switching-added noise as well, giving a bound of $p^r B_0$. Now,

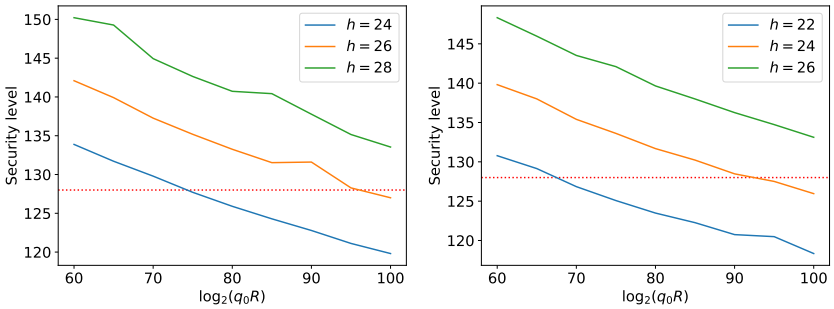
measured on the powerful basis, the scaled noise is $D_M \cdot 2\beta$ and the total noise is $D_M \cdot 2\beta + B_0 p^r$. Thus, the requirement for q_0 is $D_M \cdot 2\beta + B_0 p^r < \frac{q_0}{2\Delta}$. □

Practical Parameters Selection. In practice, we select the bootstrapping parameters in the following way. First, the Hamming weight h of the encapsulated bootstrapping key s is chosen arbitrarily. Then, the powerful bound B on the overflow parts I' and I^* is computed using Lemma 6 before the auxiliary modulus Δ is chosen. Now the value of Δ is determined in one of the two ways. It can be either set to p^t such that $p^t > 2B + 1$ (as required by digit removal algorithms), or to some integer Δ_0 that is coprime with p and satisfies $2B + 1 < \Delta_0 < \frac{p-1}{2B} - 1$ (as required by Lemma 3). Finally, the modulus $q_0 = q_{ks}$ and R are selected such that $R \geq \alpha/\beta$ and the criteria described in Lemma 7 are satisfied. The selected parameters can guarantee the correctness of the bootstrapping but do not ensure the security of the sparse bootstrapping key encapsulation. Thus, if the computed value of $q_0 R$ does not provide sufficient security (or provides excessive security), the entire process should be repeated with a larger (or smaller) h. Additionally, the main secret key s' for homomorphic computation should be generated with an appropriate Hamming weight h', which is independent of the various requirements mentioned in this section.

Table 3: Examples of Bootstrapping Parameters

Parameter Set	M	p	r	k	h	B	Δ	$\log_2(q_0)$	$\log_2(R)$
A	$32551 = 43 \cdot 757$	127	2	8	24	23	127	≥ 37.85	≥ 27.38
					26	24	127	≥ 37.85	≥ 27.38
					28	25	127	≥ 37.85	≥ 27.38
B	$45551 = 11 \cdot 41 \cdot 101$	17	4	8	22	30	17^2	≥ 41.93	≥ 29.22
					24	31	17^2	≥ 41.93	≥ 29.22
					26	32	17^2	≥ 41.93	≥ 29.22

Below, we provide a concrete illustration of the bootstrapping parameter selection. Table 3 presents two sets of parameters, including h, B, Δ, q_0, and R, selected as described above. The security estimates for different values of M, h, and $q_0 R$ are depicted in Figure 6. As Figure 6 demonstrates, the security level increases significantly with the parameter h, making it straightforward to meet security requirements by choosing a larger h. Conversely, if we fix the cyclotomic order M and the number of digits in key-switching to $L = 3$, and consider B as a constant, we can set $q_0/p^r \leq c_0$ and $R = c_1 q_0^{1/L}$ for some constants c_0, c_1 according to Lemma 6 and Lemma 7. Consequently, $q_0 R \leq c_1 c_0^{\frac{L+1}{L}} p^{r\frac{L+1}{L}} := c p^{r\frac{L+1}{L}}$ for some constant c. When combined with the concrete parameters from Table 3, it suggests that $\log_2(q_0 R) = 100$ is sufficient to support a plaintext modulus p^r of more than 30 bits, which is adequate in most scenarios.



(a) $M = 32551$ (b) $M = 45551$

Fig. 6: The impact of h and $\log_2(q_0 R)$ on the security level.

4.4 Adapting for Thin Bootstrapping

Our optimization of the digit removal process is readily adaptable for thin bootstrapping, where each slot encrypts an integer instead of an extension field (ring) element. When employing the techniques of Kim et al. [17], the thin bootstrapping procedure deviates slightly from the ordinary case as stated in Section 4.1. The major distinction is that we need to apply the CoeffToSlot transform to both $\tilde{c}$ and c''. The parameter selection remains consistent with the ordinary case and is omitted here.

SlotToCoeff. Given a ciphertext $\mathrm{Enc}_{s'}([v_0, v_1, \ldots, v_{n-1}]; p^r)$, where $v_i \in \mathbb{Z}_{p^r}$ are the integers stored in the slots, this step moves the v_i's to certain powerful basis coefficients of the encrypted plaintext polynomial using the SlotToCoeff transform. The output ciphertext of this step encrypts a polynomial whose powerful basis coefficients are all zero except those storing the slot values. Let m_0 be the corresponding plaintext polynomial.

Decryption Formula Simplification. This step is identical to the ordinary case. We can still obtain $\tilde{c} = \mathrm{Enc}_{s'}(\Delta m_0 + [q_0]_{p^{r+v_p(\Delta)}} I'; p^{r+v_p(\Delta)})$ and $(b'', a'') = \mathrm{Enc}_s(I'; \Delta)$.

Homomorphic Inner Product. Similar to the ordinary case, this step computes $c'' = b'' + a'' \cdot \mathrm{Enc}_{s'}(s; p^{r+v_p(\Delta)}) = \mathrm{Enc}_{s'}(I' + \Delta I^*; p^{r+v_p(\Delta)})$.

CoeffToSlot. The coefficients of $\mathrm{pwfl}(I' + \Delta I^*)$ encrypted in c'' are moved to the slots. Then, another linear transform is applied such that each slot contains only an integer, producing $c_0 = \mathrm{Enc}_{s'}([J'_0 + \Delta J^*_0, \ldots, J'_{n-1} + \Delta J^*_{n-1}]; p^{r+v_p(\Delta)})$, where J'_i, J^*_i are the coefficients of $\mathrm{pwfl}(I')$ and $\mathrm{pwfl}(I^*)$ and are bounded by B. On the other hand, we also need to apply these linear transforms to $\tilde{c}$ so that we can assemble them later. The ciphertext obtained from $\tilde{c}$ is $c_1 = \mathrm{Enc}_{s'}([\Delta v_0 + [q_0]_{p^{r+v_p(\Delta)}} J'_0, \ldots, \Delta v_{n-1} + [q_0]_{p^{r+v_p(\Delta)}} J'_{n-1}]; p^{r+v_p(\Delta)})$.

Digit Removal. The digit removal is performed on c_0 in a similar way to the ordinary case. The only difference is that it is performed only once instead of d times. This step outputs $\mathrm{Enc}_{s'}([J'_0, \ldots, J'_{n-1}]; p^{r+v_p(\Delta)})$.

Assembling the Parts Together. This step mirrors the process in the ordinary case. Nevertheless, note that in thin bootstrapping, we are assembling the integers in the slots, as opposed to assembling powerful basis coefficients in the ordinary case.

Remark. In the case of $\Delta = p^t$, we ignore the extracted overflow part in thin bootstrapping and directly perform digit removal on

$$[q_0]^{-1}_{p^{r+t}} c_1 = \mathrm{Enc}_{s'}([p^t [q_0]^{-1}_{p^{r+t}} v_0 + J'_0, \ldots, p^t [q_0]^{-1}_{p^{r+t}} v_{n-1} + J'_{n-1}]; p^{r+t})$$

to avoid the extra CoeffToSlot linear transform applied to c''. This choice will not harm the performance of the digit removal because the global null polynomial is usually ineffective due to its higher degree than the local null polynomials.

4.5 Combining with Existing Optimizations

Our digit removal procedure is compatible with multiple existing optimizations, mainly in homomorphic polynomial evaluation. The optimization already adopted in our implementation includes multi-poly BSGS [11], odd function optimization [8] and lazy relinearization [18].

Note that Geelen et al. used optimal-depth BSGS [3] in their work, which was initially designed for polynomial evaluation in CKKS. However, we do not use this technique, since the multiplicative depth in BGV is quite different from that in CKKS. In CKKS, both scalar and non-scalar multiplication consume the same amount of capacity per level, thus it suffices to model the cost using multiplicative depth. However, in BGV, integer multiplication, scalar multiplication, and non-scalar multiplication cost different amounts of capacity, with the non-scalar one being the most expensive. Thus, we focus on the non-scalar depth and ignore the capacity loss caused by integer multiplication when evaluating polynomials during digit removal. This choice can slightly improve the efficiency of polynomial evaluation and simplify the implementation.

Other optimizations that can be directly applied to our method include constructing low-norm polynomials by solving CVP in the null polynomial lattice [8] and fast polynomial evaluation using the Galois structure of the underlying ring [19]. However, since they either are just made public or require too much implementation effort, we decided not to include them in our analysis and implementation.

Some existing optimizations interact with our method in a non-trivial way. For example, our local null polynomials construction technique can further accelerate the function composition method developed by Geelen et al. [8]. When the digit extraction polynomial is represented as the composition of two sub-polynomials, with degrees of $(p-1)(e'-1)+1$ and $p\lceil \frac{e}{e'} \rceil$, respectively. Combined with our local null polynomial technique, their degrees can be reduced to

$\min((2B+1)\lceil\frac{e'}{t}\rceil, (p-1)(e'-1)+1)$ and $\min(p, 2B+1)\lceil\frac{e}{e'}\rceil$. For large values of p (e.g., $p > (2B+1)$), one or both of the sub-polynomials can be reduced to a lower degree.

Despite the compatibility of our digit removal with a number of existing techniques, the connection between these techniques and the overall efficiency is highly complicated, making it hard to find the optimal strategy through theoretical analysis. For example, polynomial composition will reduce the benefits brought by multi-polynomial BSGS. Also, the polynomial composition can be accelerated by increasing the value of t, which in turn affects both the degree of the local null polynomials and the capacity consumed in homomorphic operations. Moreover, many optimizations trade capacity for lower computational costs, making it more difficult to find the critical path of the digit removal procedure, which is needed by some optimizations. It might be possible, however, to find the best solution by enumerating all the possible combinations of these optimizations with a fine-grained cost model.

However, as our goal is to argue the advantage of our digit removal procedure, we choose to avoid the tedious work of finding the best combination of optimizations. Instead, we focus on simple cases that suffice to demonstrate the advantage of our method compared to previous ones. Firstly, we assume that the number of digits to extract is at most two (i.e., $2B+1 < p^2$). Since $2B+1$ is around $50 \sim 60$ in practice, this assumption holds for $p \geq 11$, which still enables a wide choice of p. Conversely, for small values of p, the degree of polynomials used in digit removal is too low for our global/local null polynomial technique to be effective. Secondly, we ignore the tradeoff between running time and capacity when constructing the local null polynomials with different values of t and always use the smallest $t = \lceil \log_p(2B+1) \rceil$.

5 Implementation

Experiment Setup. Our implementation is based on HElib [16] (commit ID `3e337a6`), which is one of the most popular libraries implementing BGV bootstrapping. We've also incorporated the sparse key encapsulation technique developed in [4] into BGV bootstrapping, as suggested by Geelen and Vercauteren [9]. For a fair comparison, our baseline also employs this technique.[8]

The experiments are conducted on a machine running Fedora 33 (Workstation Edition) equipped with a 3 GHz Intel(R) Core(TM) i9-10980XE CPU and 125GB of RAM. Consistent with previous works [15,8], the compiled program is executed in a single thread.

Parameter Selection. The parameters chosen by Halevi and Shoup [15] mostly offer a security level of 80 bits, with one parameter set providing 128 bits of security. Also, Geelen et al. [8] used parameters with a security level

[8] Note that we do not compare with the results of [8] since they do not use the sparse key encapsulation technique, which prevents a fair comparison.

Table 4: The selected parameter sets are categorized as follows: Parameter sets that share the same Roman number have identical values of p, r, M. Type-A parameter sets utilize $\Delta = p^t$, while type-B parameter sets employ $\Delta = \Delta_0$ coprime to p. Other parameter sets use HElib's original bootstrapping (HS bootstrapping). Parameters that provide 80-bit security are enclosed in braces.

ID	p^r	M	d	$\log_2(Q)$	λ'	h	λ	$\log_2(\epsilon)$	B	Δ or $p^{e-e'}$	$\log_2(q_0 R)$
I	17^4	38309	24	1462	82.5	24(12)	136.0(81.3)	-34.3	/	17^2	63.8(64.5)
I-A						24(14)	133.1(89.8)		23(18)		70.3
II	127^2	56647	45	2253	82.6	22(12)	134.1(87.3)	-33.7	/	127	69.4(60.7)
II-A						22(12)	135.4(85.7)		22(17)		66.6
III	257^2	55427	28	2176	82.8	22(12)	135.5(87.4)	-33.8	/	257	64.2(64.7)
III-A						22(12)	133.4(85.5)		22(17)		70.6
IV	8191	45193	14	1803	82.3	22(12)	130.2(83.3)	-34.0	/	8191	66.3(66.8)
IV-A						24(12)	136.7(81.9)		23(17)		72.7
IV-B						22(12)	131.8(83.8)	-33.0	22(17)	45	62.8
V	65537	50731	18	2036	82.3	22(12)	130.6(83.0)	-33.9	/	65537	74.6(75.1)
V-A						24(12)	136.8(82.3)		23(17)		81.2
V-B						22(12)	132.8(84.8)	-32.9	23(17)	47	67.1

ranging from 66 to 81 bits. In practice, using a parameter set with 128-bit security can significantly reduce the efficiency of bootstrapping or even render it impossible (i.e., bootstrapping cannot be completed successfully if it consumes more capacity than what the parameter set can provide). Thus, we opt to establish the main secret key s' with at least 80 bits of security. The concrete security level is estimated using the Lattice-Estimator [2,1](commit ID `fd4a460`).

For the encapsulated key s, we use two security levels: 80 bits and 128 bits. The 128-bit-secure version is included because the security of sparse keys is still not well understood at present and s is highly sparse under 80 bits of security (with $h = 12$). Thus, we choose to be more conservative to provide resistance against potential future attacks on sparse keys. Additionally, we also provide an implementation of the 80-bit-secure version as it offers better efficiency.

The Hamming weight of the main secret key is always set to 120 to align with HElib. The parameter k, which bounds the overflow part (see Lemma 6), is set to 8 for both original bootstrapping in HElib and our bootstrapping. This should ensure a bootstrapping failure probability less than 2^{-32} for practical parameters. Moreover, since the bound B on the overflow part is proportional to $2^{\omega(M)/2}$ (where $\omega(M)$ is the number of prime factors of the cyclotomic order M), we use M with $\omega(M) = 2$ exclusively for better performance. The candidate parameter sets are automatically generated using the `paramsx` program provided in HElib. We then select some of these candidates that meet the above criteria and have inexpensive linear transformations. The selected parameter sets are listed in Table 4.

Table 5: Benchmark results of thin bootstrapping with 128 bits of security for s. The parameter sets without any suffix use the HS bootstrapping provided by HElib. The parameter sets suffixed with '-A' use $\Delta = p^t$, where both the global and local null polynomial optimizations are available. Those suffixed with '-B' use $\Delta = \Delta_0$, where the digit removal is performed using the lifted interpolation polynomial (i.e., only the global null polynomial optimization is available).

Parameter Set ID		I	I-A	II	II-A	III	III-A	IV	IV-A	IV-B	V	V-A	V-B
Capacity (bits)	Initial	1003	1042	1573	1616	1542	1581	1253	1294	1287	1415	1462	1457
	Linear map	136	138	123	126	134	136	132	140	110	148	155	120
	Digit extract	452	332	297	282	400	311	558	274	322	808	317	385
	Remaining	408	561	1142	1200	1002	1124	552	865	847	445	972	943
Time (sec)	Linear map	12	12	29	30	32	31	20	20	36	25	25	48
	Digit extract	41	23	76	50	112	42	321	27	150	2512	33	194
	Total	54	36	107	81	145	76	342	49	187	2540	60	243
Throughput (bps)		7.59	15.42	10.69	14.73	6.92	14.84	1.62	17.51	4.52	0.175	16.10	3.89
Speedup		1x	2.03x	1x	1.38x	1x	2.15x	1x	10.8x	2.80x	1x	91.7x	22.1x

Benchmark Data and Analysis. We conduct tests on general and thin bootstrapping using the parameters given in Table 4. The *throughput* of a bootstrapping operation is calculated as the ratio of the remaining after-bootstrap capacity to the bootstrapping time, as in Geelen et al.'s work [8]. The remaining after-bootstrap capacity is defined as the after-bootstrap capacity minus the minimum capacity required for a ciphertext to be bootstrappable. For instance, the capacity needed to perform the SlotToCoeff transform in thin bootstrapping is subtracted from the after-bootstrap capacity. The test results for 128-bit-secure s are provided in Table 5 and Table 6, while those for 80-bit-secure s are included in Supplementary Material A.

As indicated in the tables, the time consumed by our digit removal procedure is significantly reduced compared to that in HElib. Our optimizations not only reduce the running time but also decrease the capacity consumed during the digit removal procedure, thereby improving the throughput from both aspects. For Type-A parameters, this improvement is due to the low-degree local null polynomials. For instance, in parameter set I-A, the second-row digit extraction polynomial has a degree of 65 and is reduced modulo a monic null polynomial of degree 15. For parameter sets II-A to V-A, the degree reductions for the first-row digit extraction polynomials are $253 \to 135$, $513 \to 135$, $8191 \to 94$ and $65537 \to 94$, respectively. For Type-B parameters (IV-B and V-B), the decrease in polynomial degrees is attributed to the lifted interpolation polynomials, which have degrees of 2023 and 2207, respectively.

For Type-A parameters, the packing transform and SlotToCoeff transform in general bootstrapping is performed with plaintext modulus p^{r+t} instead of p^r in HS bootstrapping, leading to a higher capacity consumption. However, our bootstrapping begins with a higher capacity because the elements multiplied during the inner product step are smaller compared to HS bootstrapping. Fur-

Table 6: Benchmark results of general bootstrapping with 128 bits of security for s. The parameter sets without any suffix use the HS bootstrapping provided by HElib. The parameter sets suffixed with '-A' use $\Delta = p^t$, where both the global and local null polynomial optimizations are available. Those suffixed with '-B' use $\Delta = \Delta_0$, where the digit removal is performed using the lifted interpolation polynomial (i.e., only the global null polynomial optimization is available).

Parameter Set ID		I	I-A	II	II-A	III	III-A	IV	IV-A	IV-B	V	V-A	V-B
Capacity (bits)	Initial	1003	1019	1573	1594	1542	1558	1253	1266	1287	1415	1431	1457
	Linear map	207	241	194	219	208	241	199	243	158	225	277	171
	Digit extract	446	341	298	293	400	316	559	288	329	812	331	389
	Remaining	344	434	1070	1080	927	998	484	732	791	363	820	889
Time (sec)	Linear map	93	93	285	277	319	319	113	115	113	115	112	115
	Digit extract	953	568	3176	2148	3010	1224	4535	363	2095	46088	574	3471
	Total	1046	662	3462	2427	3330	1545	4648	479	2209	46203	688	3589
Throughput (bps)		0.329	0.656	0.309	0.445	0.278	0.646	0.104	1.527	0.358	0.008	1.191	0.248
Speedup		1x	2.00x	1x	1.44x	1x	2.32x	1x	14.7x	3.44x	1x	151x	31.5x

thermore, the significant improvement of the digit removal step dominates the overall performance.

The test data also suggests that Type-B parameters, which use global null polynomials for digit removal, are generally less efficient than Type-A parameters that utilize local null polynomials. Although Type-B parameters still improve the throughput of baseline parameters, they are about 4 to 5 times less efficient than their Type-A counterparts. This is because while global null polynomials have asymptotically better degrees, local null polynomials have lower degrees within the currently considered parameter range.

Additionally, as shown in Supplementary Material A, switching from a 128-bit-secure s to an 80-bit-secure one reduces the Hamming weight from $22 \sim 24$ to $12 \sim 14$, further enhancing throughput improvement. Type-B parameters benefit most from such reduction in h, experiencing an improvement in throughput of up to 1.47 times, while Type-A parameters have their throughput increased by up to 1.24 times.

6 Conclusion

We optimize BGV bootstrapping by performing digit removal on $I^*\Delta + I' \in \mathbb{Z}_{p^r+v_p(\Delta)}$, which is independent of the message and noise in the input ciphertext. Utilizing the fact that both I^* and I' are bounded by a small integer B, we design the global null polynomial and local null polynomial optimizations to reduce the polynomial degrees during homomorphic digit removal to $\min((2B+1)^2, (2B+1)\lceil (r+v_p(\Delta))/v_p(\Delta)\rceil)$. This method is especially effective for large values of p. Experiment results show that we achieve a throughput of $1.38 \sim 151$ times that of HS bootstrapping implemented in HElib.

Acknowledgments

This work is supported by the National Key R&D Program of China (2018YFA0704701, 2020YFA0309705), Shandong Key Research and Development Program (2020ZLYS09), Natural Science Foundation of China (92267203) , the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008), and Tsinghua University Dushi Program.

References

1. Albrecht, M.R.: lattice-estimator. https://github.com/malb/lattice-estimator/ (2023)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/doi:10.1515/jmc-2015-0016, https://doi.org/10.1515/jmc-2015-0016
3. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 587–617. Springer International Publishing, Cham (2021)
4. Bossuat, J.P., Troncoso-Pastoriza, J., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security. pp. 521–541. Springer International Publishing, Cham (2022)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Trans. Comput. Theory **6**(3) (jul 2014). https://doi.org/10.1145/2633600, https://doi.org/10.1145/2633600
6. Chen, H., Han, K.: Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 315–337. Springer International Publishing, Cham (2018)
7. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144, https://eprint.iacr.org/2012/144
8. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On Polynomial Functions Modulo p^e and Faster Bootstrapping for Homomorphic Encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 257–286. Springer Nature Switzerland, Cham (2023)
9. Geelen, R., Vercauteren, F.: Bootstrapping for BGV and BFV Revisited. Journal of Cryptology **36**(2), 12 (Mar 2023). https://doi.org/10.1007/s00145-023-09454-6, https://doi.org/10.1007/s00145-023-09454-6
10. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
11. Gentry, C., Halevi, S.: Implementing Gentry's Fully-Homomorphic Encryption Scheme. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. pp. 129–148. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

12. Halevi, S.: Comment under Issue #80 of HElib. https://github.com/homenc/HElib/issues/80#issuecomment-207448286 (2016)
13. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 641–670. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
14. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481 (2020), https://eprint.iacr.org/2020/1481, https://eprint.iacr.org/2020/1481
15. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021). https://doi.org/10.1007/s00145-020-09368-7, https://doi.org/10.1007/s00145-020-09368-7
16. IBM: HElib. https://github.com/homenc/HElib/ (2023)
17. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General Bootstrapping Approach for RLWE-based Homomorphic Encryption. IEEE Transactions on Computers pp. 1–13 (2023). https://doi.org/10.1109/TC.2023.3318405
18. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 551–580. Springer International Publishing, Cham (2022)
19. Okada, H., Player, R., Pohmann, S.: Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. Cryptology ePrint Archive, Paper 2023/1304 (2023), https://eprint.iacr.org/2023/1304, https://eprint.iacr.org/2023/1304

Supplementary Material

A Benchmark Results with 80 Bits of Security for s

Table 7: Benchmark results of general bootstrapping with 80 bits of security for s.

Parameter Set ID		I	I-A	II	II-A	III	III-A	IV	IV-A	IV-B	V	V-A	V-B
Capacity (bits)	Initial	1003	1019	1580	1594	1542	1558	1253	1266	1287	1415	1431	1458
	Linear map	207	242	192	221	209	242	199	239	161	226	283	180
	Digit extract	446	340	299	257	401	276	559	286	310	812	331	359
	Remaining	334	434	1084	1114	926	1037	484	738	808	363	814	911
Time (sec)	Linear map	90	93	285	278	332	319	113	111	111	115	115	113
	Digit extract	936	552	3327	1793	3109	1043	4515	310	1514	47111	517	2450
	Total	1027	647	3614	2074	3442	1364	4629	423	1626	47227	634	2566
Efficiency (bps)		0.335	0.671	0.300	0.537	0.269	0.760	0.104	1.74	0.497	0.008	1.28	0.355
Speedup		1x	2.00x	1x	1.79x	1x	2.83x	1x	16.7x	4.76x	1x	167x	46.3x

Table 8: Benchmark results of thin bootstrapping with 80 bits of security for s.

Parameter Set ID		I	I-A	II	II-A	III	III-A	IV	IV-A	IV-B	V	V-A	V-B
Capacity (bits)	Initial	1003	1042	1580	1617	1542	1581	1253	1294	1287	1415	1462	1458
	Linear map	136	138	123	126	134	136	132	141	110	148	155	121
	Digit extract	449	332	296	247	400	271	558	273	321	808	317	354
	Remaining	411	561	1156	1234	1002	1164	552	865	848	445	972	975
Time (sec)	Linear map	12	12	29	30	32	31	20	20	36	25	26	47
	Digit extract	43	23	75	43	112	37	325	24	109	2609	28	140
	Total	55	36	106	75	145	70	347	46	147	2635	56	189
Efficiency (bps)		7.44	15.6	10.9	16.5	6.93	16.6	1.59	19.0	5.76	0.169	17.4	5.16
Speedup		1x	2.09x	1x	1.51x	1x	2.40x	1x	11.9x	3.62x	1x	103x	30.5x

B Exploiting Local Null Polynomials in HS/GV Bootstrapping

The overflow part extraction technique [17] is necessary to construct low-degree global null polynomials, where the entire input to digit extraction is required to be sparse (i.e., both higher and lower digits should be small). However, low-degree local null polynomials can be constructed without extracting the overflow

part, because it only requires the lower digits to be small and has no requirements for the higher digits. In this section, we show how to port the local null polynomial optimization to HS and GV bootstrapping without the technique of [17].

B.1 Local Null Polynomials in HS Bootstrapping.

First, we introduce the algorithmic procedure of HS bootstrapping. The HS bootstrapping is parameterized by two parameters e and e'. Given a BGV ciphertext with plaintext modulus p^r, they first modulus-switch the ciphertext to the special modulus $q = p^e + 1$, where $e > r$. Denote the obtained ciphertext by $c = (b, a) \in \mathcal{R}^2$, where b, a should be q-reduced on the powerful basis. Every polynomial used here is assumed to be in its powerful basis representation. Note that

$$b + as = m + p^r\epsilon + Iq = (m + I) + p^r\epsilon + p^eI, \tag{13}$$

where $m \in \mathcal{R}_{p^r}$ is the plaintext polynomial, ϵ is the RLWE noise and $m + p^r\epsilon$ should be q-reduced to ensure the correctness of decryption. The polynomial $I = \lfloor \frac{b+as}{q} \rceil$ can be understood as the overflow part of $b + as$ modulo q.

Observe that $b + as \equiv m + I \bmod p^r$, so homomorphically computing $b + a \cdot Enc(s; p^r)$ results in a ciphertext encrypting $\mathrm{Enc}(m + I; p^r)$ with large homomorphic capacity. To remove the effect of I, it suffices to obtain the encryption of I under plaintext modulus p^r, or equivalently $I\langle r - 1, 0\rangle_p$.

Observe that if $m+I+p^r\epsilon$ is p^e-reduced, the value represented by the highest r digits in $[b + as]_{p^{e+r}}$ is exactly $[I]_{p^r}$. Thus, denoting $b + as$ as m', we have the simplified decryption formula

$$m \equiv m'\langle r - 1, 0\rangle - m'\langle e + r - 1, e\rangle \bmod p^r. \tag{14}$$

The BGV decryption function is now simplified to computing $m' = b + as$ modulo p^{e+r} homomorphically, extracting the lowest r and the highest r digits, and finally computing their difference.

Halevi and Shoup noticed that, by adding multiples of q to b and a, we can make both b and a divisible by $p^{e'}$ for some integer e'. Denote the b and a after that by $b' = b + q\Delta_b$ and $a' = b + q\Delta_a$, then $m' = b' + a's = m + p^r\epsilon + I' + p^eI'$ is also divisible by $p^{e'}$, i.e., $m'\langle e' - 1, 0\rangle = 0$, where $I' = I + \Delta_b + \Delta_a s$. Note that $m+I'+p^r\epsilon$ also needs to be p^e-reduced to ensure the correctness of the decryption formula. Let $m'' = \frac{b'}{p^{e'}} + \frac{a'}{p^{e'}}s = \frac{m'}{p^{e'}}$, this optimization further simplifies the decryption formula to

$$m \equiv p^{e'}m''\langle r - 1 - e', 0\rangle - m''\langle e + r - e' - 1, e - e'\rangle \bmod p^r. \tag{15}$$

Additionally, if $e' \geq r$, we have $m \equiv -m''\langle e+r-e'-1, e-e'\rangle \bmod p^r$, which means extracting the highest r digits of m'' suffices to recover m.

Here we assume $e' \geq r$ because the increase in noise caused by a larger e' can be compensated by choosing a larger e, as we will see later. In this case, to extract $m''\langle e + r - e' - 1, e - e'\rangle$ with low-degree local null polynomials, it is

crucial to bound the size of the lower digits $(m + I' + p^r\epsilon)/p^{e'}$. Equation (12) from [15] provides a high-probability bound on these lower digits as

$$B_{HS} = |m + I' + p^r\epsilon|^{\text{pwfl}}/p^{e'} \leq B_0(p^{e'}(1 + 1/q) + 2p^r + 1 + 1/q^2)/p^{e'}. \quad (16)$$

This bound should also satisfy $B_{HS} \leq p^{e-e'}/2$ for correctness (i.e., $m + I' + p^r\epsilon$ is p^e-reduced).

Now, the lowest $e' - e$ digits of m'' is exactly $(m + I' + p^r\epsilon)/p^{e'}$. By choosing e' to be sufficiently large and adjusting e for correctness requirements, the bound on $m''\langle e - e' - 1, 0\rangle$ (i.e., B_{HS}) can be made arbitrarily close to B. For example, one may set $p^{e'} \geq 2B_0(2p^r + 1 + 1/q^2)$ such that $B_{HS} \leq B_0(1 + 1/q) + 1/2 \approx \lfloor B_0 \rceil = B$. This indicates there exists a local null polynomial of degree no more than $\lceil \frac{e-e'+r}{e-e'}(2B_{HS} + 1)\rceil \approx \lceil \frac{e-e'+r}{e-e'}(2B + 1)\rceil$ using Lemma 4, which is similar to the results in Section 4.2.

B.2 Local Null Polynomials in GV Bootstrapping.

Recently, Geelen and Vercauteren provided a unified decryption formula for both BGV and BFV, which has a simpler structure than the result of Halevi and Shoup. Similar to the HS bootstrapping, the GV bootstrapping also modulus-switches the input ciphertext to the special modulus q, satisfying $\gcd(q, p) = 1$. Denote the ciphertext modulo q by $c = (b, a) \in \mathcal{R}_q^2$, which satisfies $[b + as]_q = m + p^r\epsilon$. If we multiply c by p^{e-r} modulo q, the resulting ciphertext $(b', a') = ([p^{e-r}b]_q, [p^{e-r}a]_q) \in R^2$ satisfies

$$b' + a's = p^{e-r}m + p^e\epsilon + I'q. \quad (17)$$

Let $b'' = [b']_{p^e} \cdot [q]_{p^e}^{-1}$ and $a'' = [a']_{p^e} \cdot [q]_{p^e}^{-1}$, we have

$$b'' + a''s \equiv (b' + a's) \cdot [q]_{p^e}^{-1} \equiv [q]_{p^e}^{-1}p^{e-r}m + I' \bmod p^e. \quad (18)$$

If $|I'|^{\text{pwfl}} = |\frac{b'+a's-(p^{e-r}m+p^e\epsilon)}{q}|^{\text{pwfl}} < \frac{p^{e-r}}{2}$, the value represented by the highest r digits in $[b'' + a''s]_{p^e}$ is exactly $[q^{-1}m]_{p^r}$. Let $m'' = b'' + a''s$, the decryption formula is then simplified to

$$m \equiv qm''\langle e - 1, e - r\rangle \bmod p^r. \quad (19)$$

Now it only remains to bound $|I'|^{\text{pwfl}}$ to construct the low-degree local null polynomials. By increasing q, the contribution of the term $\frac{p^{e-r}m+p^e\epsilon}{q}$ can be made arbitrarily small, leading to $|I'|^{\text{pwfl}} \approx |\frac{b'+a's}{q}|^{\text{pwfl}} = B_0$.

Our decryption formula simplification is a special case of GV bootstrapping if we ignore the adoption of methods in [17]. Recall that in Section 4.1, we require $|\Delta(m + p^r\epsilon)|^{\text{pwfl}} < q_0/2$ for $\Delta = p^t$. This requirement corresponds to $|p^{e-r}m+p^e\epsilon|^{\text{pwfl}} < q/2$ in GV bootstrapping by substituting $e = r+t$ and $q_0 = q$. In this case, $p^{e-r}m + p^e\epsilon = [b' + a's]_q$, implying $I' = (b' + a's - [b' + a's]_q)/q = \lfloor (b' + a's)/q \rceil$ and $|I'|^{\text{pwfl}} = \lfloor B_0 \rceil = B$.