Laurent Polynomial-Based Linear Transformations for Improved Functional Bootstrapping

San Ling[1], Benjamin Hong Meng Tan[2], Huaxiong Wang[1] and Allen Siwei Yang[1,2]

[1] School of Physical & Mathematical Sciences, Nanyang Technological University, Singapore
lingsan@ntu.edu.sg, hxwang@ntu.edu.sg, yang0788@e.ntu.edu.sg
[2] Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore
benjamin_tan@i2r.a-star.edu.sg

Abstract. Following Gentry's seminal work (STOC 2009), Fully Homomorphic Encryption (FHE) has made significant advancements and can even evaluate functions in the bootstrapping process, called functional bootstrapping. Recently, Liu and Wang (ASIACRYPT 2023) proposed a new approach to functional bootstrapping, which bootstrapped ciphertexts in 7ms amortized time. Their methods packed the secret key of the TFHE cryptosystem into a ciphertext of the BFV cryptosystem, followed by performing functional bootstrapping of TFHE within BFV. However, while this yields high amortized efficiency, it faces high latency and computational complexity of $\mathcal{O}(\sqrt{t})$ ciphertext-ciphertext multiplications due to use of large BFV plaintext primes that serve as the TFHE ciphertext modulus, $t = 65537$, to maximize SIMD slots.

In this work, we adapt their techniques to achieve lower latency functional bootstrapping by relaxing the requirement for prime BFV plaintext modulus to prime powers, $t = p^r$. We first introduce an improved linear transformation stage, multiplying Laurent Polynomial packed secret key and ciphertexts, a_{ij} and sk_j, evaluating a $\mathbb{Z}_{p^r}$ linear map. With this, we reduce the number of operations needed to evaluate the linear phase of bootstrapping. Finally, we generalize their functional bootstrapping procedure from plaintext space $\mathbb{Z}_t$ to $\mathbb{Z}_{p^r}$ via leveraging the digit extraction algorithm, achieving a theoretical complexity of $\mathcal{O}(r^2\sqrt{p})$ ciphertext-ciphertext multiplications. Additionally, we enable a multi-valued bootstrapping scheme that permits the evaluation of multiple functions over a shared input. To the best of our knowledge, this is the first demonstration of such a method for TFHE ciphertexts that relies predominantly on BFV-based techniques.

In our experiments, we achieve overall runtimes as low as 49.873s, representing latency reductions of at least 26×, while noting a 19× slowdown in amortized performance.

Keywords: Fully Homomorphic Encryption, Linear Transformation, Functional Bootstrapping.

1 Introduction

Fully Homomorphic Encryption (FHE) is a type of encryption scheme which allows for the computation of arbitrary functions over encrypted data, without undermining security. It has a wide range of applications, including privacy-preserving machine learning [34, 38], private set intersection (PSI) [25] and multiparty computation (MPC) [11].

FHE was first attained in the seminal work by Gentry [19]. One would first begin with a somewhat homomorphic encryption scheme. Here, homomorphic operations would be performed, accruing noise in the ciphertexts with each operation. Eventually, the noise would exceed the threshold for correct decryption in the ciphertexts and be rendered undecryptable. To overcome this, the somewhat homomorphic scheme would be converted into a fully homomorphic scheme via a process known as bootstrapping. Bootstrapping performs a homomorphic evaluation of the decryption circuit, refreshing the noise and thereby allowing for more operations to be performed. Due to the high cost of bootstrapping, numerous works seek to improve upon the efficiency of bootstrapping.

The current state of FHE comprises primarily of three different types of schemes, each with their own characteristics. Brakerski-Gentry-Vaikuntanathan (BGV) [4] and Brakerski-Fan-Vercauteren (BFV) [3, 14] schemes make up the first type. These schemes support a Single Instruction Multiple Data (SIMD) plaintext structure. This is achieved by packing messages into a ring element using the Chinese Remainder Theorem (CRT) and then encrypting it as a Ring Learning with Errors (RLWE) ciphertext. This allows for evaluation of circuits with multiple inputs but shallower depth, allowing for many messages to be packed and operated on in parallel. As a result, this grants the advantage of lower amortized time at the expense of higher latency of homomorphic operations.

The FHEW [13] and TFHE [9] schemes comprise the second type. As opposed to the batching of multiple messages into a ciphertext, these schemes focus on low latency bootstrapping on single messages. Specifically, a single bit is encrypted in a learning with errors (LWE) ciphertext and bootstrapping is performed in conjunction with a NAND gate evaluation between two such bits. Extensions of the basic FHEW and TFHE schemes for larger plaintext spaces introduced functional bootstrapping, where arbitrary functions on the FHEW/TFHE plaintext space can be evaluated alongside bootstrapping. This supports circuits of larger multiplicative depth, offering the advantage of lower latency but at the expense of increased amortized time. The third variant would be that of CKKS [7], which supports approximate complex arithmetic as well as the SIMD structure.

1.1 Related work

Several works have attempted to perform bootstrapping of one scheme using another. This was first introduced in [2] and expanded upon in [33], where scheme switching can let one take advantage of the strengths of the different schemes. Recently, [31] proposed a novel bootstrapping framework which bootstrapped TFHE ciphertexts using BFV. Here, the TFHE LWE secret key and multiple

TFHE LWE ciphertexts decryptable under the same secret key are packed into one BGV RLWE ciphertext and plaintexts respectively. A homomorphic linear transformation is then evaluated to perform the homomorphic inner product of bootstrapping. This is followed by a homomorphic lookup table evaluation, enabled by translating the look-up table into a polynomial via interpolation and evaluating said polynomial. Finally, modulus switching, key switching and ciphertext extraction were performed. This achieved functional bootstrapping in less than 7ms amortized time, with $\tilde{O}(1)$ polynomial multiplications.

Another recent work [27] bootstrapped BFV ciphertexts using the CKKS scheme. Here, the CKKS scheme was leveraged to homomorphically extract the noise from BFV ciphertexts, then subtracted from the initial ciphertext to obtain the desired bootstrapped result. This too achieved improved performance, with an amortized bootstrapping time of 1.08ms, approximately 30× faster over the current state of the art [17] for BFV bootstrapping.

However, while the amortized timings of [31] are low, their methods suffer from high latency due to the use of large plaintext primes. Furthermore, while [27] provides improved timings, the techniques used make it challenging to incorporate functional evaluation as part of the bootstrapping pipeline. Therefore, motivated by these, we attempt to develop a low latency variant of [31], which performs functional bootstrapping via utilization of different schemes.

In addition to efforts to bootstrap one scheme using another, several concurrent studies have concentrated on improving the performance of functional bootstrapping by considering its non-linear, function evaluation aspects. Their methods represent further directions and optimizations that could be used in conjunction with the techniques we introduce.

A new notion of relaxed functional bootstrapping was introduced in [32]. The correctness requirement of regular bootstrapping is relaxed to a subset of the plaintext space, allowing circuit-specific functional bootstrapping constructions. Various constructions of lookup tables exploiting this notion were described, yielding 1-2× speed up of regular BGV/BFV bootstrapping and a 3× and 6× improvement of latency and throughput respectively over the prior work of [31].

In [29], a novel functional bootstrapping framework for BFV and CKKS was introduced, with the focus being on the functional component. Here, a modulus switch to pack the ciphertexts into a larger ciphertext modulus is performed. Following this, the arbitrary lookup table component is then evaluated. The primary idea behind the arbitrary lookup table evaluation was to first decompose the function to be evaluated into multiple Heaviside functions. Each of these Heaviside functions could then be further recursively decomposed into products of polynomials and sub-lookup tables. These were then evaluated, providing enhanced functionality for an arbitrary lookup table between plaintext spaces $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ where p is prime and $r > s$. In their experiments, functional bootstrapping latencies of 47.3s and 172.1s were achieved, with amortized timings of 2956.25ms and 10756.25ms. However, compared to our results for similar security, their amortized timings are worse, being 1.40× to 5.11× slower, although they do achieve 1.57× to 5.70× lower latency. Furthermore, their experiments

were only performed for the evaluation of delta and sign functions, which require only 1 Heaviside function evaluation. Larger arbitrary lookup tables would be more computationally intensive to evaluate, requiring multiple Heaviside function evaluations. We leave it to future work to explore if their novel techniques could be integrated in our bootstrapping method.

Lastly, multi-valued functional bootstrapping is a bootstrapping technique that enables the simultaneous evaluation of multiple arbitrary functions on a given input. This concept was initially introduced for TFHE in [5], where multiple lookup tables (LUTs) were evaluated within a single bootstrapping operation. Subsequent improvements were made in [20] through tree-based methods and in [10] via multi-output programmable bootstrapping. More recently, further advancements in multi-valued functional bootstrapping have been achieved in [28], [35], and [30]. Despite this progress, to the best of our knowledge, no prior work has explored the use of primarily BFV-based bootstrapping techniques to realize multi-valued functionality within TFHE. Our work therefore also represents the first approach to achieve multi-valued functional bootstrapping for TFHE ciphertexts via BFV techniques.

1.2 Contributions

In this paper, we propose a new framework for bootstrapping, extending the work of [31] to support TFHE ciphertext modulus and BGV/BFV plaintext spaces that are prime powers. Similar to [31], multiple LWE ciphertexts $(\overrightarrow{a_i}, b_i)$ sharing the same secret key are bootstrapped. However, in this work, an improved linear transformation algorithm is utilized. This is due to the use of smaller plaintext primes for the RLWE plaintext space, which decreases the number of plaintext slots for packing the secret key entries. If the homomorphic linear transformation proposed in [31] were utilized, multiple calls to the linear transformation followed by a summation over the outputs of the linear transformations would be performed. This contrasts with the single call of the transformation in their original work. Our work improves upon this by utilizing a Laurent polynomial encoding followed by a projection map, which takes advantage of the fact that any $\mathbb{Z}_{p^r}$ linear map can be expressed as a sum of Frobenius powers. In addition to this, we provide a generalized version of their linear transformation algorithm for the case where the dimension of the LWE ciphertext is not exact multiple of the number of slots.

Besides that, we generalize the arbitrary homomorphic function evaluation proposed in [31] to the case of prime power plaintext spaces. To do so, the digit extraction algorithm of [22] is applied. As described in [31], their method for arbitrary homomorphic function evaluation requires that the RLWE plaintext space used is prime. In utilizing small primes however, the RLWE plaintext space used in our work takes the form of a prime power. To overcome this, we propose using the digit extraction algorithm to perform homomorphic prime division, followed by the corresponding arbitrary homomorphic function evaluation. This reduces the plaintext space to a prime, allowing for arbitrary homomorphic function evaluation.

Additionally, we introduce what is, to our knowledge, the first instance of multi-valued functional bootstrapping for TFHE ciphertexts that utilizes primarily BFV-based techniques. In the framework of [31], utilizing multi-valued functional bootstrapping would involve repeated calls to the steps after the linear transformation, for each distinct function. Due to the already high latency of the procedure, this would be impractical. By enabling the use of prime power BGV/BFV plaintext spaces, this can be performed with lower latency. In particular, for a single function evaluated using the framework of [31], our proposed framework allows for the evaluation of at least 84 distinct functions in the time it takes to bootstrap a single set of ciphertexts in [31].

We implement our work for prime powers $p^r = 127^3$ and 257^3 with cyclotomic degree $N_{\text{rlwe}} = 32768$, achieving overall runtimes of 49.873s and 243.967s respectively at about 80-bit security. This represents a latency improvement of at least $26.95\times$ and $5.51\times$ compared to [31]. We note however, that there is a trade-off in amortized time that arises from our methods, evidenced by a slowdown of at most $19\times$ or $46.46\times$ depending on the parameters. Thus, we present the first realization of the RLWE prime-power plaintext variant of [31], with significantly improved latency and support for multi-valued TFHE bootstrapping.

1.3 Technical Overview

To begin, LWE secret key entries are packed into an RLWE ciphertext. However, rather than packing the secret key directly into the plaintext slots as per [31], the entries are encoded as the coefficients of a Laurent polynomial with negative powers. This polynomial is then packed repeatedly into the slots of a ciphertext. From here, the different LWE $\overrightarrow{a_i}$s are each encoded into separate Laurent polynomials with positive powers and each Laurent polynomial packed into plaintext slots. The packed Laurent polynomials containing $\overrightarrow{a_i}$ are each multiplied with the Laurent polynomial containing the secret key via plaintext-ciphertext multiplication and summed accordingly. This results in the desired $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ in the first coefficient of the polynomial in each ith slot, due to the exponents cancelling out in the convolution. By evaluating a projection map to the first coefficient, $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ can be obtained in each slot, without the other exponents of the convolution. $b_i - \langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ is then computed slot-wise. This completes the initial linear transformation of homomorphic decryption.

Following this, digit extraction is performed. This maps $\frac{p^r}{p} \cdot m_i + e_i \mapsto m_i$ for each slot, with p^r being the RLWE plaintext modulus and LWE ciphertext modulus. This completes the base bootstrapping procedure. Polynomial evaluation corresponding to the arbitrary homomorphic function evaluation is then performed, evaluating the desired lookup table slot-wise on messages m_i. The RLWE ciphertext containing the evaluated m_is is then modulus switched to the original LWE ciphertext modulus p^r. It is also key switched to contain the desired LWE secret key. Finally, RLWE to LWE extraction is performed, obtaining the functionally bootstrapped LWE ciphertexts.

In the case of multi-valued functional bootstrapping, the same bootstrapping procedure is applied, with the steps from the polynomial evaluation onwards being repeated for each distinct function. The linear transformation and digit extraction is only computed once. Furthermore, for the polynomial evaluation component, computation of the powers X^i as part of the Paterson-Stockmeyer algorithm [36] need also only to be performed once. That is, only the second half of the polynomial evaluation procedure is applied for each distinct function to be evaluated.

2 Preliminaries

2.1 Notation

For any vector $\overrightarrow{a}$, let $\overrightarrow{a}[j]$ or a_j denote the jth entry of the vector. For any polynomial a, let $a[j]$ or a_j denote the jth coefficient of the polynomial. For any matrix A, let $A[i][j]$ denote the entry in the ith row and jth entry. All indexing begins at index 0. Let $\langle \cdot, \cdot \rangle$ denote the inner product between two vectors. Let $\mathcal{U}(\cdot)$ denote the uniform distribution. All secret keys used are assumed to follow a ternary distribution. That is, for secret key $\overrightarrow{sk} \in \mathbb{Z}_q^n$, $\overrightarrow{sk}[j] \in \{-1, 0, 1\}, 0 \leq j \leq n-1$ and for secret key $sk \in \mathcal{R}_Q$, $sk[j] \in \{-1, 0, 1\}, 0 \leq j \leq N-1$, where N is the ring dimension.

2.2 Fully Homomorphic Encryption

To date, practical fully homomorphic encryption schemes are constructed with lattices and have their security based on the decisional learning with errors (LWE) and ring learning with errors (RLWE) problems.

Decisional LWE. Let $n_{\text{lwe}}, q_{\text{lwe}}$ be positive integers, $\mathcal{D}$ be a secret key distribution and χ be an error distribution over $\mathbb{Z}$. The decisional LWE problem asks to distinguish between the distributions of $(\overrightarrow{a}, b = \langle \overrightarrow{a}, \overrightarrow{sk} \rangle + e) \in (\mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}} \times \mathbb{Z}_{q_{\text{lwe}}})$, where $\overrightarrow{a} \leftarrow \mathcal{U}(\mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}}), e \leftarrow \chi, \overrightarrow{sk} \leftarrow \mathcal{D}$, and $(\overrightarrow{a}, b) \leftarrow \mathcal{U}(\mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}} \times \mathbb{Z}_{q_{\text{lwe}}})$.

Decisional RLWE. Let $N_{\text{rlwe}}, Q_{\text{rlwe}}$ be positive integers, $\mathcal{R}_{Q_{\text{rlwe}}}$ a ring of dimension N_{rlwe}, $\mathcal{D}$ be a secret key distribution and χ be an error distribution over $\mathcal{R}_{Q_{\text{rlwe}}}$. The decisional RLWE problem asks to distinguish between the distributions of $(a, b = a \cdot sk + e) \in \mathcal{R}_{Q_{\text{rlwe}}}^2$, where $a \leftarrow \mathcal{U}(\mathcal{R}_{Q_{\text{rlwe}}}), e \leftarrow \chi, sk \leftarrow \mathcal{D}$, and $(a, b) \leftarrow \mathcal{U}(\mathcal{R}_{Q_{\text{rlwe}}}^2)$.

2.2.1 FHEW/TFHE

For FHEW [12]/TFHE [9], LWE-based ciphertexts are the focus of the bootstrapping techniques introduced in [31] and the techniques briefly discussed here. Let t_{lwe} be the plaintext modulus, q_{lwe} be the ciphertext modulus, $\mathcal{D}, \chi$ be the secret key and error distributions.

Let $m \in \mathbb{Z}_{t_{\text{lwe}}}$ be the message, $\alpha = \lfloor \frac{q_{\text{lwe}}}{t_{\text{lwe}}} \rfloor$ be the scaling factor, $\overrightarrow{sk} \leftarrow \mathcal{D}$ be the secret key and $e \leftarrow \chi$ be the error. Sample $\overrightarrow{a} \leftarrow \mathcal{U}(\mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}})$. The LWE ciphertext is of form $(\overrightarrow{a}, b) \in \mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}+1}$, satisfying

$$b = \alpha \cdot m + \langle \overrightarrow{a}, \overrightarrow{sk} \rangle + e \pmod{q_{\text{lwe}}}$$

with $|e| < \lfloor \frac{q_{\text{lwe}}}{2t_{\text{lwe}}} \rfloor$. Decryption is computed via $\left\lfloor \frac{b - \langle \overrightarrow{a}, \overrightarrow{sk} \rangle \pmod{q_{\text{lwe}}}}{\alpha} \right\rceil$.

The main difference between FHEW/TFHE from other FHE schemes such as BGV and BFV is the use of a separate homomorphic accumulator for bootstrapping LWE ciphertexts instead of relying on the original scheme's homomorphic properties to evaluate its own decryption circuit.

2.2.2 BFV/BGV

Some of the first practical FHE schemes, the BFV [3,14] and BGV [4] schemes, primarily differ on how plaintexts are encoded in the RLWE ciphertexts. In [31], LWE ciphertexts were packed and their LWE secret key was encrypted into a BFV RLWE ciphertext, before using the BFV scheme's homomorphic property to evaluate the LWE decryption circuit on the encrypted LWE secret key. For our work, BGV is used, but both are presented here for context.

Let $m \in \mathbb{Z}$. Let $\mathcal{R}_{Q_{\text{rlwe}}} = \mathbb{Z}_{Q_{\text{rlwe}}}[X]/\Phi_m(X)$ be the RLWE ciphertext space, $\mathcal{R}_{t_{\text{rlwe}}} = \mathbb{Z}_{t_{\text{rlwe}}}[X]/\Phi_m(X)$ be the RLWE plaintext space where $Q_{\text{rlwe}}, t_{\text{rlwe}} \in \mathbb{Z}$. Additionally, we assume $t_{\text{rlwe}} \nmid N_{\text{rlwe}} = \phi(m)$ is prime or a prime power.

Encryption. Let $\mathcal{D}, \chi$ be the secret key and error distributions. Let $m \in \mathcal{R}_{t_{\text{rlwe}}}$ be the plaintext, $sk \leftarrow \mathcal{D}$ be the secret key and $e \leftarrow \chi$ be the error. Sample $a \leftarrow \mathcal{U}(\mathcal{R}_{Q_{\text{rlwe}}})$. An RLWE ciphertext has form $(a, b) \in \mathcal{R}_{Q_{\text{rlwe}}}^2$ satisfying:

- For BFV, the message is encoded in the higher bits, with $\alpha = \lfloor \frac{Q_{\text{rlwe}}}{t_{\text{rlwe}}} \rfloor$,

  $$b = \alpha \cdot m + a \cdot sk + e.$$

  Decryption is performed via $\left\lfloor \frac{b - a \cdot sk \pmod{Q_{\text{rlwe}}}}{\alpha} \right\rceil$.
- For BGV, the message is encoded in the lower bits.

  $$b = m + a \cdot sk + t_{\text{rlwe}} \cdot e$$

  Decryption is performed via $b - a \cdot sk \bmod t_{\text{rlwe}}$.

It is possible to switch between BGV and BFV encodings for a given ciphertext via multiplication by a constant, see Appendix A of [1] for the details.

Operations. For secret key sk, let the encryption of a plaintext m be $Enc_{sk}(m)$ and decryption of a ciphertext ct be $Dec_{sk}(ct)$.

- Addition: $Dec_{sk}(Add(ct_1, ct_2)) = m_1 + m_2$ for $Dec_{sk}(ct_i) = m_i$.
- Multiplication: $Dec_{sk}(Mult(ct_1, ct_2)) = m_1 \cdot m_2$ for $Dec_{sk}(ct_i) = m_i$.
- Polynomial Evaluation: $Dec_{sk}(Eval(ct, f)) = f(m)$ for polynomial f.

- Modulus Switching: $Modswitch(ct, Q'_{\text{rlwe}}) = \frac{Q'_{\text{rlwe}}}{Q_{\text{rlwe}}} \cdot ct$.
- Key Switching: $Dec_{sk'}(KS(ct, sk', sk)) = m$ where $ct = Enc_{sk}(m)$.
- Rotation: $Dec_{sk}(Rot(ct, rtk, i)[j]) = m[j - i \pmod l]$, where rtk contains the corresponding rotation keys.

Plaintext Structure. Per [39], let p be the characteristic of t_{rlwe}, we have $\Phi_m(X) \equiv F_0(X) \cdot \ldots \cdot F_{\ell-1}(X) \bmod p$, for some $\ell \mid N_{\text{rlwe}} = \phi(m)$. This gives the isomorphism below from the Chinese Remainder Theorem and Hensel's lifting lemma

$$\mathcal{R}_{t_{\text{rlwe}}} \cong \bigoplus_{i=0}^{\ell-1} \mathbb{Z}_{t_{\text{rlwe}}}[X]/F'_i(X)$$

.

Here, each $F'_i(X) \bmod t_{\text{rlwe}}$ has degree d and is lifted from $F_i(X)$ if t_{rlwe} is a prime power and is $F_i(X)$ if t_{rlwe} is prime. These $\mathbb{Z}_{t_{\text{rlwe}}}[X]/F'_i(X)$ are all isomorphic to one another, i.e. for $E = \mathbb{Z}_{t_{\text{rlwe}}}[X]/F_0(X)$, we have $E \cong \mathbb{Z}_{t_{\text{rlwe}}}[X]/F_i(X)$ for all $0 \leq i \leq \ell - 1$. This is known as the Single Instruction Multiple Data (SIMD) slot structure, where a vector of length ℓ with degree d polynomials in each entry, can be packed into the BGV/BFV plaintext space. Data within the slots can also be rotated among the SIMD slots through automorphisms of the form $X \mapsto X^\kappa$ for $\kappa \in \mathbb{Z}_m^*/\langle p \rangle$. Furthermore, when m is a power of two, Lemma 1 characterises the form of the factors $F'_i(X)$.

Lemma 1 *[16] Let $m \geq 4$ be a power of two, then each factor $F'_i(X)$ is of the shape $F'_i(X) = X^d + a_i \cdot X^{d/2} + b_i$ where $a_i = 0$ if $p \equiv 1 \pmod 4$ and a_i is some constant if $p \equiv 3 \pmod 4$.*

Laurent Polynomial Encoding. To exploit the available space within each SIMD slot, polynomials in $\mathbb{Z}_{t_{\text{rlwe}}}[X]$ with degree at most $d - 1$, [6] further proposed the Laurent Polynomial encoding. Here, integer values are decomposed with respect to a base and the digits are then encoded as the coefficients of the Laurent Polynomial. In this work, we only consider integers in $\mathbb{Z}_{t_{rlwe}}$ and directly encode them into the coefficients without further decomposition.

Definition 1 *[6] A Laurent Polynomial $a(X) \in \mathbb{Z}[X^{\pm 1}]$ is an integral polynomial of form $a(X) = a_{\tilde{l}} X^{\tilde{l}} + \ldots + a_{\tilde{m}-1} X^{\tilde{m}-1} + a_{\tilde{m}} X^{\tilde{m}}$, where $a_i \in \mathbb{Z}, a_{\tilde{l}}, a_{\tilde{m}} \neq 0$ and $\tilde{l} \leq \tilde{m}$.*

Crucially, for correctness when using this encoding, it is required for $\tilde{m} - \tilde{l} + 1 \leq d$.

$\mathbb{Z}_{t_{\text{rlwe}}}$ Linear Functions. Fundamental to the structure of BGV is the use of $\mathbb{Z}_{t_{\text{rlwe}}}$ linear maps. This allows such maps to be expressed as a linear combination of Frobenius powers and is formally stated below in Definition 2 and Lemma 2.

Definition 2 *[15] A function $L \colon \mathbf{R}_{t_{\text{rlwe}}} \to \mathbf{R}_{t_{\text{rlwe}}}$ is $\mathbb{Z}_{t_{\text{rlwe}}}$-linear if $L(\boldsymbol{\alpha} + \boldsymbol{\alpha'}) = L(\boldsymbol{\alpha}) + L(\boldsymbol{\alpha'})$ and $L(c \cdot \boldsymbol{\alpha}) = c \cdot L(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha}, \boldsymbol{\alpha'} \in \mathbf{R}_{t_{\text{rlwe}}}$ and $c \in \mathbb{Z}_{t_{\text{rlwe}}}$.*

Lemma 2 *[15, 22, 37] Any $\mathbb{Z}_{t_{\text{rlwe}}}$ linear function $L : E \to E$ can be expressed as a linear combination of Frobenius powers. i.e. $L(\eta) = \sum_{f=0}^{d-1} \theta_f \sigma_E^f(\eta)$, where $\theta_f \in E$ are constants and $\sigma_E \colon \zeta \mapsto \zeta^p$ is the Frobenius map on E. When L has image $\mathbb{Z}_{t_{\text{rlwe}}} \subset E$, $\theta_f = \sigma_E^f(\theta_0)$.*

2.3 Functional Bootstrapping with the BGV/BFV Scheme

We present the main ideas of [31]. This takes as input ℓ-many LWE ciphertexts $\{(\overrightarrow{a_i}, b_i)\}_{i=0}^{\ell-1}$ under the same secret key $\overrightarrow{sk} = (sk_0, sk_1, \ldots, sk_{n_{\text{lwe}}-1})$, which is the number of slots available from the BGV/BFV parameters used. For their RLWE parameters, plaintext modulus t_{rlwe} is set to be a large prime p such that $N_{\text{rlwe}} = \ell$ and $d = 1$. The LWE ciphertext modulus is set such that $q_{\text{lwe}} = t_{\text{rlwe}}$.

The LWE decryption process can be thought of as consisting of three phases. The first is linear, computing $b - \langle \overrightarrow{a}, \overrightarrow{sk} \rangle$. Next, a non-linear operation which was dubbed "Division, Round and Map" or DRaM in [31] is performed . Finally, the resulting functionally bootstrapped plaintexts is recovered through key and modulus switching and RLWE-to-LWE extractions.

For high efficiency, they proposed to use the BFV scheme with $N_{\text{rlwe}} = 32768$ and a mid-sized prime $t_{\text{rlwe}} = 65537$, that can bootstrap 32768 ciphertexts simultaneously. This is illustrated in Figure 1.

Homomorphic Linear Transformation. Intuitively, we treat the ℓ $\overrightarrow{a_i}$'s of the LWE ciphertexts as a matrix $A \in \mathbb{Z}_{t_{\text{rlwe}}}^{N_{\text{rlwe}} \times n_{\text{lwe}}}$ with rows $\overrightarrow{a_i}$ and $\overrightarrow{b} = [b_i]_{i=0}^{\ell-1}$ and secret key $\overrightarrow{sk} = [sk_j]_{j=0}^{n_{\text{lwe}}}$ as a column vector. Since $n_{\text{lwe}} \mid \ell$, we pack $\lfloor \frac{\ell}{n_{\text{lwe}}} \rfloor$ copies of $\overrightarrow{sk}$, in the form $\overrightarrow{sk}$, $(sk_0, \ldots, sk_{n_{\text{lwe}}-1}, \ldots, sk_0, \ldots, sk_{n_{\text{lwe}}-1}) \in \mathbb{Z}_{t_{\text{rlwe}}}^{\ell}$ as $bfvct$ for optimal performance. Rotation keys to enable arbitrary rotations between slots, labelled as rtk, are precomputed.

Lines 3 to 45 of Algorithm 1 result in a ciphertext (res_0) that holds the desired $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ in its i-th slot. It is a baby-step giant-step variant, utilizing n_{lwe} plaintext-ciphertext multiplications and $2\sqrt{n_{\text{lwe}}}$ rotations. From here, res_0 is subtracted from b, with b being an encoding of $(b_0, b_1, \ldots, b_{N_{\text{rlwe}}-1})$ in the slots. This computes $b_i - \langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ homomorphically, yielding $\alpha \cdot m_i + e_i$ in each slot.

Homomorphic Divide, Round and Map (DRaM). Let $f : \mathbb{Z}_{t_{\text{lwe}}} \to \mathbb{Z}_{t_{\text{lwe}}}$ be any function to be evaluated on LWE messages m_i. The lookup table in the RLWE plaintext space can then be set as $L(X) = f(\lfloor \frac{X}{\alpha} \rfloor) \cdot \alpha$, where $\alpha = \lfloor \frac{q_{\text{lwe}}}{t_{\text{lwe}}} \rfloor$.

Lemma 3 *[31] For any prime t_{lwe}, let $f : \mathbb{Z}_{t_{\text{lwe}}} \to \mathbb{Z}_{t_{\text{lwe}}}$ be a function over $\mathbb{Z}_{t_{\text{lwe}}}$ and define $\tilde{f}(X) = f(0) - \sum_{i=1}^{p-1} X^i \sum_{a=0}^{p-1} f(a) a^{p-1-i}$. Then, it holds that for any $x \in \mathbb{Z}_{t_{\text{lwe}}}$, $f(x) = \tilde{f}(x)$.*

Using Lemma 3, this can be applied by evaluating

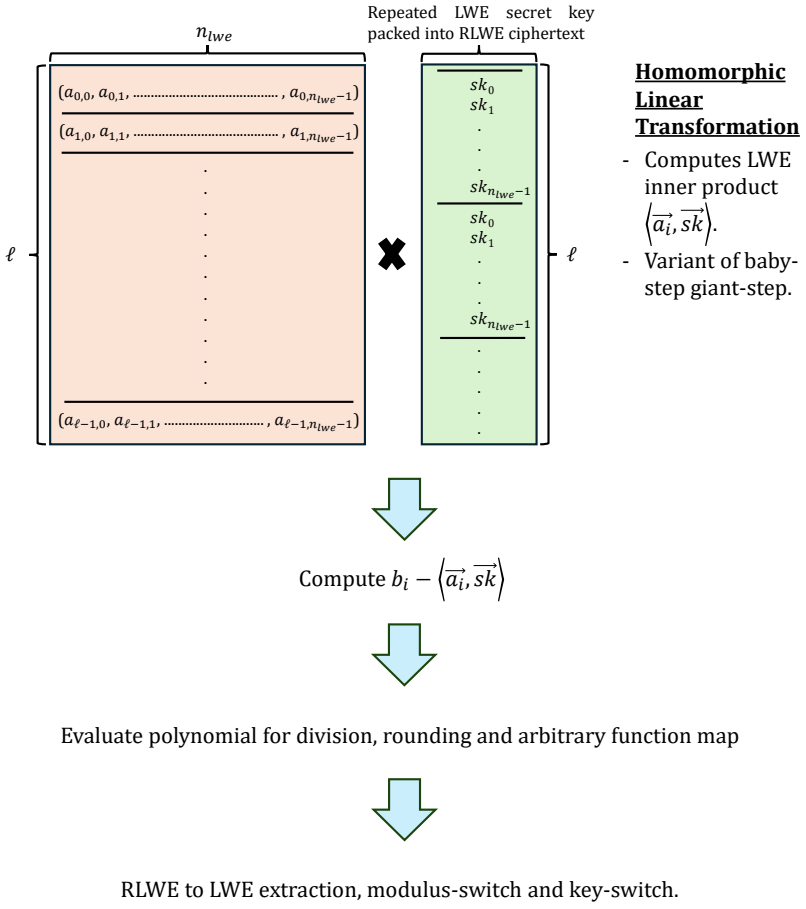

Fig. 1: Overview of the bootstrapping framework of [31].

$$\tilde{f}(X) = L(0) - \sum_{i=1}^{t_{\mathrm{rlwe}}-1} X^i \sum_{a=0}^{t_{\mathrm{rlwe}}-1} L(a) a^{t_{\mathrm{rlwe}}-1-i}$$

on the result of the homomorphic linear transformation.

Polynomial evaluation itself is performed utilizing the Paterson-Stockmeyer algorithm [36]. In brief, for a polynomial of form $\tilde{f}(X) = \sum_{i=0}^{t_{\mathrm{rlwe}}-1} l_i X^i$ to be evaluated, set $\tilde{t} = \left\lceil \sqrt{t_{\mathrm{rlwe}} - 1} \right\rceil$. Powers x^i are then computed for $1 \leq i \leq \tilde{t}$, requiring $\tilde{t} - 1$ ciphertext-ciphertext multiplications. Denote and compute $\tilde{f}_i(x) = \sum_{j=0}^{\tilde{t}-1} l_{i\tilde{t}+j} x^j$, utilizing $\tilde{t}$ plaintext-ciphertext multiplications for each i.

Algorithm 1 Homomorphic Linear Transformation of [31]

procedure LT(rtk, A, $bfvct$, b),
▷ Rotation keys, LWE $\overrightarrow{a_i}$'s, RLWE ctxt, LWE b_i's
  $rt \leftarrow \sqrt{n_{\text{lwe}}}$
  for $0 \leq i \leq rt - 1$ **do**
    $bfvct_{rot_i} \leftarrow Rot(bfvct, rtk, -(i+1) \cdot rt)$ ▷ Storing rotations of $bfvct$
    $res_i \leftarrow Enc_{sk}(0)$
  for $1 \leq k \leq rt$ **do**
    for $1 \leq i \leq rt$ **do**
      for $0 \leq j \leq N_{\text{rlwe}} - 1$ **do**
        $ind_{ct} \leftarrow j - k + 1 \pmod{N_{\text{rlwe}}}$
        $ind_a \leftarrow j + i \cdot rt \pmod{n_{\text{lwe}}}$
        $tmp[j] \leftarrow A[ind_{ct}][ind_a]$
      $c \leftarrow Mult(tmp, bfvct_{rot_{i-1}})$
      $res_{k-1} \leftarrow Add(res_{k-1}, c)$
  for $1 \leq i \leq rt - 1$ **do**
    $c \leftarrow Rot(res_{rt-i}, rtk, -1)$
    $res_{rt-i-1} \leftarrow Add(res_{rt-i}, c)$
return $b - res_0$

The polynomial can then be expressed as

$$\tilde{f}(x) = \tilde{f}_0(x) + x^{\tilde{t}} \left(\tilde{f}_1(x) + x^{\tilde{t}} \left(\dots \left(\tilde{f}_{\left\lfloor \frac{t_{\text{rlwe}}-1}{\tilde{t}} \right\rfloor - 1}(x) + x^{\tilde{t}} \tilde{f}_{\left\lfloor \frac{t_{\text{rlwe}}-1}{\tilde{t}} \right\rfloor}(x) \right) \dots \right) \right)$$

which therefore requires approximately t_{rlwe} plaintext-ciphertext multiplications and $O(\sqrt{t_{\text{rlwe}}})$ ciphertext-ciphertext multiplications to compute overall.

Slots to Coefficients. The decoding map is homomorphically evaluated on the ciphertext, mapping slot entries $L(m_i)$ to RLWE plaintext $m = \sum_{i=0}^{N_{\text{rlwe}}-1} L(m_i) X^i$. This map is performed via $m \mapsto m(\zeta^h)_{h \in S}$ where $S \in \mathbb{Z}$ is a set of representatives for $\mathbb{Z}_m^* / \langle p \rangle$.

Modulus Switching and Key Switching. Prior to extracting the LWE ciphertexts from the RLWE ciphertext, key switching and modulus switching are first performed. The RLWE secret key $sk \in \mathcal{R}_{Q_{\text{rlwe}}}$ is key-switched to $sk' = \sum_{i=0}^{N_{\text{rlwe}}-1} s'_i X^i \in \mathcal{R}_{Q_{\text{rlwe}}}$ where $s'_i = \overrightarrow{sk}[i]$ for $i \leq n_{\text{lwe}} - 1$ and $s'_i = 0$ otherwise. Modulus switching is then performed, changing RLWE ciphertext modulus from Q_{rlwe} to q_{lwe}.

RLWE to LWE Extraction. Finally, ℓ LWE ciphertexts are extracted from the RLWE ciphertext.

Let the RLWE ciphertext be $(a = \sum_{i=0}^{N_{\text{rlwe}}-1} a_i X^i, b = \sum_{i=0}^{N_{\text{rlwe}}-1} b_i X^i) \in \mathcal{R}_{Q_{\text{rlwe}}}^2$. LWE ciphertexts $(\overrightarrow{a_i'}, b'_i) \in \mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}+1}$ are obtained, for $0 \leq i \leq N_{\text{rlwe}} - 1$,

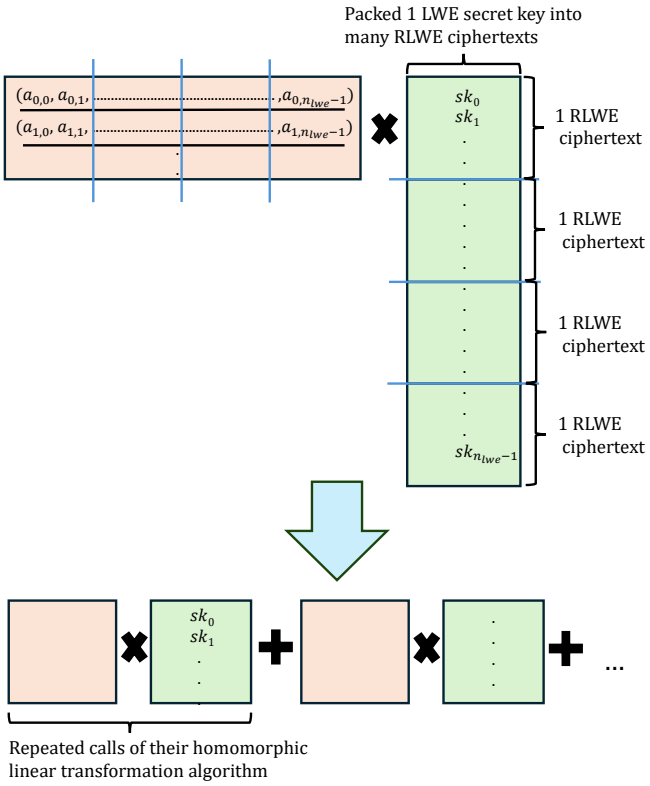

Fig. 2: Linear transformation of [31] in the small plaintext primes case.

where $a'_{i,j} = a_{i-j}$ for $j \leq i$, $a'_{i,j} = -a_{N_{\text{rlwe}}-j+i}$ for $j > i$, and $b'_i = b_i$, $0 \leq j \leq n_{\text{lwe}} - 1$.

Let the RLWE secret key be $sk = \sum_{i=0}^{N_{\text{rlwe}}-1} sk_i X^i$ after key switching. LWE secret key $\overrightarrow{sk'} \in \mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}}$ is then obtained by setting $sk'[j] = sk[j]$ for $0 \leq j \leq n_{\text{lwe}} - 1$.

3 Improved Linear Transformation

3.1 Limitations of Current Linear Transformation

In the case of small plaintext primes p, as d is the order of $p \pmod m$, the degree d of $F_i(X)$ generally increases. As $\phi(m) = \ell \cdot d$, the number of slots ℓ becomes small. The decreased slot count prevents the packing of all sk_js of $\overrightarrow{sk}$ into one BGV plaintext polynomial. As a result, to utilize the linear transformation of [31], the LWE secret key is split over multiple encryptions. This requires that the

LWE a_{ij}s are also split and results in repeated calls of the linear transformation. At the end, these are summed.

In more detail, with $\ell < n_{\text{lwe}}$ and n_{lwe} denoting the LWE dimension, $\lceil \frac{n_{\text{lwe}}}{\ell} \rceil$ RLWE ciphertexts $ct_0, ct_1, \ldots, ct_{\lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1}$ are obtained, where ct_k contains

$$(sk_{k \cdot \ell}, sk_{k \cdot \ell + 1}, \ldots, sk_{(k+1) \cdot \ell - 1})$$

for $k < \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$ and

$$(sk_{k \cdot \ell}, sk_{k \cdot \ell + 1}, \ldots, sk_{n_{\text{lwe}} - 1}, 0, \ldots, 0)$$

for $k = \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$, in slots.

Similarly, the $a_{i,j}$s are split into $\lceil \frac{n_{\text{lwe}}}{\ell} \rceil$ blocks, where $0 \leq i \leq \ell - 1$. Block k, where $0 \leq k \leq \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$, consists of ℓ plaintexts, with $ptxt_i$ containing

$$(a_{i,k \cdot \ell}, a_{i,k \cdot \ell + 1}, \ldots, a_{i,k \cdot \ell + \ell - 1})$$

for $k < \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$ and

$$(a_{i,k \cdot \ell}, a_{i,k \cdot \ell + 1}, \ldots, a_{i,n_{\text{lwe}} - 1}, 0, \ldots, 0)$$

for $k = \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$ in the plaintext slots.

From here, the linear transformation is applied $\lceil \frac{n_{\text{lwe}}}{\ell} \rceil$ times between Block k and ct_k for $0 \leq k \leq \lceil \frac{n_{\text{lwe}}}{\ell} \rceil - 1$ and results are summed. This results in $\lceil \frac{n_{\text{lwe}}}{\ell} \rceil \cdot \ell$ plaintext-ciphertext multiplications and $\lceil \frac{n_{\text{lwe}}}{\ell} \rceil \cdot 2\sqrt{\ell}$ rotations being used. This is illustrated in Figure 2.

Packed Matrix-Vector Multiplication for Vectors with Length co-prime to SIMD Slot Count. Besides the multiple calls to Algorithm 1 above, their linear transformation algorithm itself, Algorithm 1, is restricted to the case where $n_{\text{lwe}} \mid \ell$. In general, the dimension (n_{lwe}) of the LWE ciphertext need not be an exact multiple of the number of slots available through the SIMD technique. Previous work [21, 26, 31, 33] considered the case where all dimensions involved are divisors of the SIMD slot count and so did not require zero-padding and handling when "tall" matrices did not fit exactly. The requirement that $n_{\text{lwe}} \mid \ell$ imposes additional constraints on the parameters available for use. We therefore extend Algorithm 1 of [31] to handle homomorphic linear transformation that use "tall" matrices whose row comprise the entire SIMD slot count but have number of columns co-prime to the SIMD slot count. This is described in Algorithm 2.

Intuitively, zero-padding at the end of the ciphertext causes the very last packed copy of the input vector to not "see" a complete copy of the input vector with the usual set of rotations; e.g. if there were 3 zero-padded slots at the end with the input vector having length 5, then the first 3 slots in the last copy of the input vector will not see any of the entries that are before it, e.g. $(A, B, C, D, E, 0, 0, 0)$ will generate $(B, C, D, E, 0, 0, 0, A)$, $(C, D, E, 0, 0, 0, A, B)$, $(D, E, 0, 0, 0, A, B, C)$, $(E, 0, 0, 0, A, B, C, D)$ and $(0, 0, 0, A, B, C, D, E)$ without rotating beyond the vector length.

Algorithm 2 Homomorphic Linear Transformation for "Irregular" "Tall" Matrices

procedure IRREGULARLT(rtk, $A \in \mathbb{Z}_t^{r \times c}$, in, b),
▷ Rotation keys, LWE $\overrightarrow{a_i}$'s, RLWE ctxt, LWE b_i's
  $n_{\text{rot}} \leftarrow c + (\ell \bmod c)$ ▷ Largest rotation required
  $bs \leftarrow 1 \ll \lceil \log_2 \sqrt{n_{\text{rot}}} \rceil$
  $gs \leftarrow \lceil \frac{n_{\text{rot}}}{bs} \rceil$
  for $0 \le j < gs$ **do**
    $in_{rot_j} \leftarrow Rot(in, rtk, -j \cdot bs)$ ▷ Storing giant-step rotations of in
  for $0 \le i < bs$ **do**
    $res_i \leftarrow Enc_{sk}(0)$ ▷ Storing intermediate results for baby-step rotation
  for $0 \le k < \ell$ **do**
    $tmp[k] \leftarrow 0$ ▷ Start with a zero vector
  for $0 \le i < bs$ **do**
    for $0 \le j < gs$ **do**
      $rot \leftarrow j \cdot bs + i$
      $i_{\text{extra\_rotation}} \leftarrow (rot \ge c)$
      for $0 \le k < \ell$ **do**
        $i_{\text{in\_pad}} \leftarrow (k \ge \ell - \ell \bmod c - j \cdot bs)$ && $(k < (\ell - j \cdot bs))$
        $i_{\text{overflow}} \leftarrow (k < i)$ || $(k \ge \ell - j \cdot bs)$
        $ind_{ct} \leftarrow (\ell + k - i) \bmod \ell$
        $ind_a \leftarrow ((k + j \cdot bs) \bmod \ell) \bmod c$
        if i_{overflow} is True **then**
          $ind_{rev} \leftarrow \ell - ind_{ct}$
        else
          $ind_{rev} \leftarrow c$
        $i_{\text{out\_of\_range}} \leftarrow (ind_{ct} \ge r)$
        $i_{\text{rev\_ind\_done}} \leftarrow (ind_{rev} \le (rot - c))$
        $i_{\text{zero\_unused}} \leftarrow (i_{\text{extra\_rotation}}$ && $!i_{\text{overflow}})$
        $i_{\text{zero\_wrapped}} \leftarrow (i_{\text{extra\_rotation}}$ && i_{overflow} && $i_{\text{rev\_ind\_done}})$
        if $i_{\text{out\_of\_range}}$ || $i_{\text{in\_pad}}$ || $i_{\text{zero\_unused}}$ || $i_{\text{zero\_wrapped}}$ **then**
          $tmp[k] \leftarrow 0$
        else
          $tmp[k] \leftarrow A[ind_{ct}][ind_a]$
      $c \leftarrow Mult(tmp, in_{rot_j})$
      $res_i \leftarrow Add(res_i, c)$
  for $1 \le i < bs$ **do**
    $c \leftarrow Rot(res_i, rtk, -i)$
    $res_0 \leftarrow Add(res_0, c)$
return $b - res_0$

This causes two issues. First, to ensure that each slot of the output ciphertext will be covered by every entry of the input, we need to rotate by more than the length of the encrypted vector. Secondly, rotating by more than the length of the vector causes some segments of the SIMD slots to see more than one copy of certain entries, requiring us to apply masks appropriately. With each rotation we do, for the weight vector (tmp in Algorithm 2), we set slots that are not supposed to contribute for that rotation to zero. For rotations that are less than the vector length, we set to zero any slots that do not end up within the output vector. For rotations that are greater than the vector length, we set to zero any slot that do not belong in the wrap-around region. Besides that, we also set to zero any slot in the wrap-around region that had already seen the complete input vector, by looking at their "reversed slot index", which is their slot index in the SIMD vector starting from 1 from the back. Roughly speaking, the last k slots of the SIMD vector will be complete once we rotate by $c + k - 1$, where c is the input vector length. Algorithm 2 therefore roughly performs baby-step giant-step on twice the input vector length, resulting in complexity $O(\sqrt{|in|})$ where $|in|$ denotes the length of the input vector.

3.2 Laurent Polynomial Encoding

The difference between the linear transformation applied to the parameters of [31] and the one applied in this context would be that here, multiple calls to the linear transformation have to be applied due to there being fewer slots.

To improve upon this, in the cases where N_{rlwe} is a power of 2, we aim to better utilize the plaintext slots. To do so, we pack $a_{i,j}$s and sk_js into the coefficients of a slot polynomial via the Laurent Polynomial Encoding as stated in Definition 1. Note that here, LWE ciphertext modulus q_{lwe} is selected to be of form t_{lwe}^r. That is, $q_{\text{lwe}} = t_{\text{lwe}}^r = t_{\text{rlwe}}$ where t_{lwe} is prime. Since a common prime is used, we denote $p^r = t_{\text{lwe}}^r = t_{\text{rlwe}}$ for brevity.

To further define the encoding, we differentiate between the cases where $p \equiv 1 \pmod 4$ and $p \equiv 3 \pmod 4$. This is due to the slot polynomial modulus $F_i'(X)$ affecting the correctness of multiplication with this encoding, which will be further elaborated on in Section 3.4. This results in Laurent Polynomials of degree d being utilized for $p \equiv 1 \pmod 4$ and Laurent Polynomials of degree $\frac{d}{2}$ being utilized for $p \equiv 3 \pmod 4$. It is also assumed that $d \mid n_{\text{lwe}}$ here.

3.2.1 Case $p \equiv 1 \pmod 4$: The LWE secret key $\overrightarrow{sk}$ is encoded over $\frac{n_{\text{lwe}}}{d}$ RLWE plaintexts. For a given plaintext, each slot will contain the same polynomial. For plaintext k' in slot i, secret key entries are packed in the slot as the coefficients of the polynomial

$$\sum_{j=0}^{d-1} sk_{j+k' \cdot d} X^{-j} \pmod{F_i'(X)}$$

where $0 \leq k' \leq \frac{n_{\text{lwe}}}{d} - 1$, $0 \leq i \leq \ell - 1$. This is illustrated in Figure 3. Each plaintext k' is then encrypted as ciphertext k.

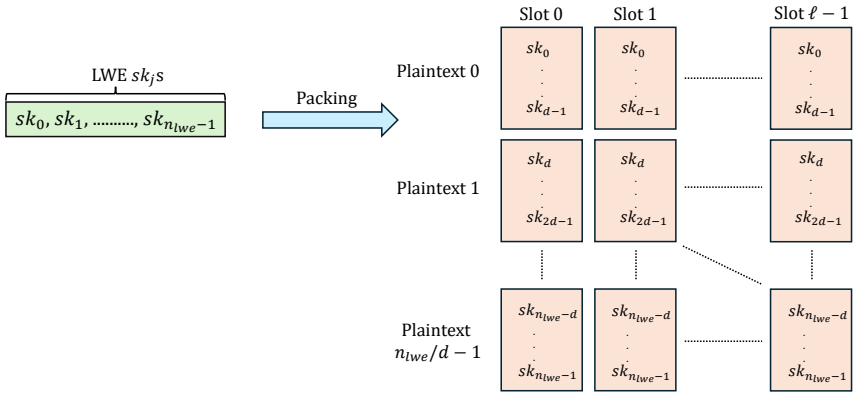

Fig. 3: Packing of LWE sk_j entries for Laurent polynomial-based linear transformation.

The LWE $a_{i,j}s$ are encoded over $\frac{n_{\text{lwe}}}{d}$ RLWE plaintexts. For plaintext k in slot i, $a_{i,j}$s are packed in the slot as the coefficients of the polynomial

$$\sum_{j=0}^{d-1} a_{i,j+k\cdot d} X^j \pmod{F_i'(X)}$$

where $0 \leq k \leq \frac{n_{\text{lwe}}}{d} - 1$, $0 \leq i \leq \ell - 1$. This is illustrated in Figure 4.

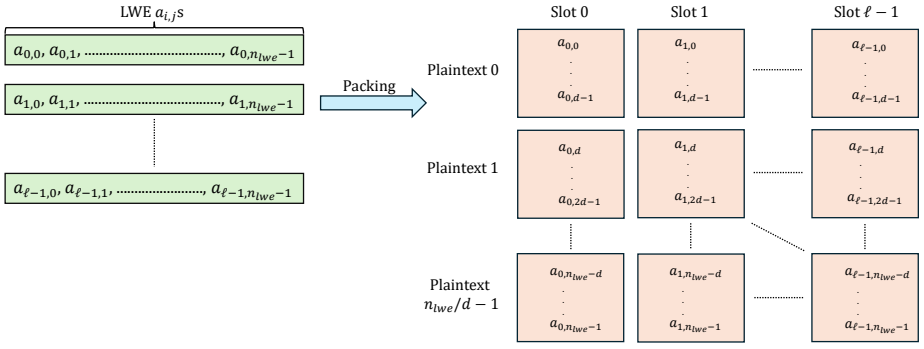

Fig. 4: Packing of LWE $a_{i,j}$ entries for Laurent polynomial-based linear transformation.

3.2.2 Case $p \equiv 3 \pmod 4$: The LWE secret key is encoded over $\frac{n_{\text{lwe}}}{d/2}$ RLWE plaintexts. For a given plaintext, each slot will contain the same polynomial. For plaintext k', in slot i, secret key entries are packed in the slot as the coefficients of the polynomial

$$\sum_{j=0}^{d/2-1} sk_{j+k' \cdot (d/2)} X^{-j} \pmod{F_i'(X)}$$

where $0 \leq k' \leq \frac{n_{\text{lwe}}}{d/2} - 1$, $0 \leq i \leq \ell - 1$. Each plaintext k' is encrypted as ciphertext k.

The LWE $a_{i,j}s$ are encoded over $\frac{n_{\text{lwe}}}{d/2}$ RLWE plaintexts. For plaintext k, in slot i, $a_{i,j}$s are packed in the slot as the coefficients of the polynomial

$$\sum_{j=0}^{d/2-1} a_{i,j+k \cdot (d/2)} X^{j} \pmod{F_i'(X)}$$

where $0 \leq k \leq \frac{n}{d/2} - 1$, $0 \leq i \leq \ell - 1$.

From here, let $F_i'(X) = g(X)X + F_i'(0)$. Under assumption $F_i'(0) \equiv 1 \pmod{p^r}$, we obtain $X^{-1} \equiv -g(X)F_i'(0)^{-1} \pmod{F_i'(X)}$. Following [6], we utilize the maps $X \mapsto X$ and $X^{-1} \mapsto -g(X)F_i'(0)^{-1}$ to obtain the corresponding degree $d-1$ or degree $d/2-1$ slot polynomials, packing the Laurent Polynomials into the slots.

3.3 Multiplication and $\mathbb{Z}_{p^r}$ Linear Map

Following the encoding, for each k, plaintext k and ciphertext k are multiplied. These results are then summed. The multiplications and summation provide the desired inner product of LWE decryption $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$, in the first coefficient of the polynomial in slot i. This results in $\frac{n_{\text{lwe}}}{d}$ plaintext-ciphertext multiplications being used if $p \equiv 1 \pmod 4$ and $\frac{n_{\text{lwe}}}{d/2}$ plaintext-ciphertext multiplications being used if $p \equiv 3 \pmod 4$.

The map $\pi_0 : E \to E$ defined as $\sum_{j=0}^{d-1} a_j x^j \mapsto a_0$ is a $\mathbb{Z}_{p^r}$-*linear* function as per Definition 2. This map is then performed across all slots via an evaluation of a linear combination of Frobenius powers by Lemma 2.

To apply the map, the product with the first constant $\theta_0 \cdot \eta$ is first computed homomorphically via one plaintext-ciphertext multiplication. The slot-wise trace map τ is then applied, mapping $\theta_0 \cdot \eta \mapsto \sum_{j=0}^{d-1} \sigma_E^j(\theta_0 \cdot \eta) = \sum_{f=0}^{d-1} \theta_f \sigma_E^f(\eta)$ as described in [23]. The trace map is computed using a special case variant of the trace algorithm provided in [21], where d is a power-of-two. The variant is formally provided as Algorithm 3. This utilizes $\log d$ many rotations and ciphertext-ciphertext additions. Applying this map, we finally obtain the desired LWE inner product in the slots.

Finally we subtract this from a plaintext containing b_i in each slot i. That is, we compute $b_i - \langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ homomorphically in the slots. This completes the linear portion of the homomorphic decryption.

Algorithm 3 *Log_2Trace*

procedure LOG2TRACE(*ctxt*, *d*)
  result ← *ctxt*
  for $0 \leq i \leq \log_2 d - 1$ **do**
    temp ← *result*
    index ← 2^i
    Apply Frobenius Automorphism to power of *index* to temp.
    result ← *result* + *temp*
return *result*

3.4 Choice of Packing Degree/Correctness

It was previously discussed in Section 3.2 that only Laurent Polynomials up to degree $\pm d/2$ are utilized for $p \equiv 3 \pmod 4$ and degree $\pm d$ for $p \equiv 1 \pmod 4$. This is due to the difference in the structure of $F_i'(X)$.

From Lemma 1 and Section 3.2,

$$X^{-1} \equiv -(X^{d-1} + a_i X^{d/2-1}) b_i^{-1} \pmod{F_i'(X)}$$

.

Hence for powers of form X^j and $X^{-(d/2+j)}$,

$$\sum_{j=0}^{d/2-1} a_{i,j} X^j \cdot sk_{d/2+j} X^{-(d/2+j)}$$
$$\equiv \sum_{j=0}^{d/2-1} -a_{i,j} sk_{d/2+j} (b_i^{-1} a_i + b_i^{-1} X^{d/2}) \pmod{F_i'(X)}$$

.

There is therefore an additional coefficient $\sum_{j=0}^{d/2-1} -a_{i,j} sk_{d/2+j} (b_i^{-1} a_i)$ added to the constant term from the convolution, when performing the multiplication.

In the case of $p \equiv 1 \pmod 4$, by Lemma 1, this additional coefficient is 0 and hence powers up to $\pm d$ can be utilized. In the case of $p \equiv 3 \pmod 4$, by Lemma 1, this additional coefficient is not necessarily zero, and therefore prevents having only the LWE inner product $\sum_{j=0}^{d-1} a_{i,j} sk_j$ in the first coefficient. It is trivial to see that X^j and $X^{-(d/2+j)}$, for $0 \leq j \leq d/2 - 1$, are the only pairs of powers from the convolution whose product has a non-zero term in the first coefficient. Powers of X, $\geq d/2$ and $\leq -d/2$ are hence not utilized in the packing for this case.

The details and correctness of the multiplication and $\mathbb{Z}_{p^r}$ linear maps for both cases in slot i are thus as follows.

3.4.1 Case $p \equiv 1 \pmod 4$:

1. Multiplication between each of the $\frac{n_{\text{lwe}}}{d}$ plaintext-ciphertext pairs.

$$\left(\sum_{j=0}^{d-1} sk_{j+k\cdot d}X^{-j}\right)\cdot\left(\sum_{j=0}^{d-1} a_{i,j+k\cdot d}X^{j}\right)$$
$$=\left(\sum_{j=0}^{d-1} a_{i,j+k\cdot d}sk_{j+k\cdot d}\right)+X(\ldots)\pmod{F_i'(X)}$$

2. Summation over all the $\frac{n_{\text{lwe}}}{d}$ products.

$$\sum_{k=0}^{\frac{n_{\text{lwe}}}{d}-1}\left(\left(\sum_{j=0}^{d-1} a_{i,j+k\cdot d}sk_{j+k\cdot d}\right)+X(\ldots)\right)$$
$$=\left(\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\right)+X(\ldots)\pmod{F_i'(X)}$$

3. Applying $\mathbb{Z}_{p^r}$-*linear* map π_0 to obtain the LWE inner product in the slots.

$$\left(\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\right)+X(\ldots)\mapsto\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\pmod{F_i'(X)}$$

3.4.2 Case $p \equiv 3 \pmod 4$:

1. Multiplication between each of the $\frac{n_{\text{lwe}}}{d/2}$ plaintext-ciphertext pairs.

$$\left(\sum_{j=0}^{d/2-1} sk_{j+k\cdot(d/2)}X^{-j}\right)\cdot\left(\sum_{j=0}^{d/2-1} a_{i,j+k\cdot(d/2)}X^{j}\right)$$
$$=\left(\sum_{j=0}^{d/2-1} a_{i,j+k\cdot(d/2)}sk_{j+k\cdot(d/2)}\right)+X(\ldots)\pmod{F_i'(X)}$$

2. Summation over all the $\frac{n_{\text{lwe}}}{d/2}$ products.

$$\sum_{k=0}^{\frac{n_{\text{lwe}}}{d/2}-1}\left(\left(\sum_{j=0}^{d/2-1} a_{i,j+k\cdot(d/2)}sk_{j+k\cdot(d/2)}\right)+X(\ldots)\right)$$
$$=\left(\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\right)+X(\ldots)\pmod{F_i'(X)}$$

3. Applying $\mathbb{Z}_{p^r}$-*linear* map π_0 to obtain the LWE inner product in the slots.

$$\left(\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\right)+X(\ldots)\mapsto\sum_{j=0}^{n_{\text{lwe}}-1} a_{i,j}sk_j\pmod{F_i'(X)}$$

3.5 Analysis

As discussed in Section 3.3, multiplication and summation of the Laurent Polynomial encoded plaintexts and ciphertexts utilize $\frac{n_{\mathrm{lwe}}}{d}$ plaintext-ciphertext multiplications if $p \equiv 1 \pmod 4$ and $\frac{n_{\mathrm{lwe}}}{d/2}$ plaintext-ciphertext multiplications if $p \equiv 3 \pmod 4$. In the case of the projection map, one plaintext-ciphertext multiplication is applied, followed by $\log d$ rotations which dominate the cost of the trace map.

The total cost of the Laurent polynomial-based linear transformation is therefore $\frac{n_{\mathrm{lwe}}}{cd} + 1$ plaintext-ciphertext multiplications and $\log d$ rotations, where $c \in \{\frac{1}{2}, 1\}$.

Comparison. We conduct a comparison of our Laurent polynomial-based linear transformation to the linear transformation of [31] in the case of small p such that $\ell < n_{\mathrm{lwe}}$.

As discussed in Section 3.1, their linear transformation utilizes $\lceil \frac{n_{\mathrm{lwe}}}{\ell} \rceil \cdot \ell \approx n_{\mathrm{lwe}}$ plaintext-ciphertext multiplications. Our Laurent polynomial-based linear transformation therefore outperforms theirs in terms of plaintext-ciphertext multiplication by either a factor of d if $p \equiv 1 \pmod 4$ or $d/2$ if $p \equiv 3 \pmod 4$.

As discussed in Section 3.1, their linear transformation utilizes $\lceil \frac{n_{\mathrm{lwe}}}{\ell} \rceil \cdot 2\sqrt{\ell}$ rotations. Under the assumption that $n_{\mathrm{lwe}} > \frac{\sqrt{\ell}}{2} \log d$, it follows that $\log d < \lceil \frac{n_{\mathrm{lwe}}}{\ell} \rceil \cdot 2\sqrt{\ell}$. This assumption easily holds in the case of small p as ℓ is the dominating factor on the right hand side of the inequality and a smaller p corresponds to a fewer number of slots ℓ. Our Laurent polynomial-based linear transformation therefore outperforms theirs in terms of number of rotations utilized. As our linear transformation outperforms the linear transformation of [31] both in terms of plaintext-ciphertext multiplications and rotations, it achieves better performance overall in the case of small p.

For comparison with the original parameters used in the linear transformation of [31], from Section 2.3, note that their linear transformation requires n_{lwe} plaintext-ciphertext multiplications and $2\sqrt{n_{\mathrm{lwe}}}$ rotations.

Similar to the analysis above, our Laurent polynomial-based linear transformation outperforms theirs in terms of plaintext-ciphertext multiplication by either a factor of d if $p \equiv 1 \pmod 4$ or $d/2$ if $p \equiv 3 \pmod 4$. In the case of rotations, we have that $\log d < 2\sqrt{n_{\mathrm{lwe}}}$ easily holds with our choice of parameters. This is further verified in Section 7. As our linear transformation in the small p case outperforms the linear transformation of [31] under their original parameters both in terms of plaintext-ciphertext multiplications and rotations, it outperforms theirs overall.

4 Non-Linear Components

In the bootstrapping framework of [31], upon completing the homomorphic linear transformation, Lemma 3 is directly applied to perform the LUT evaluation.

However, Lemma 3 requires the RLWE plaintext space to be a prime as opposed to a prime power, which is the RLWE plaintext space utilized in our framework. To overcome this, digit extraction is performed before applying Lemma 3 to first set the plaintext space from p^r to p. This section discusses utilizing the digit extraction algorithm alongside Lemma 3 to obtain the desired LUT evaluation in the RLWE prime power plaintext space.

4.1 Components

Digit Extraction. After completing the inner product and subtraction of homomorphic decryption, digit extraction is then utilized to apply homomorphic decryption and rounding by p^v, where $v = e - r'$. The digit extraction algorithm is formally stated in Section 6 of [18]. This is applied to enable the use of Lemma 3 for the functional component later, which requires that the plaintext space is prime rather than a prime power.

Let $\sum_{i=0}^{e-1} w_i p^i$ be the base-p digit decomposition of any given $w \in \mathbb{Z}_{p^e}$, where $|w_i| \leq p/2$. For odd p, the digit extraction procedure homomorphically computes

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

.

In the context of our bootstrapping procedure, v is set to $r-1$, with $e = r$, $r' = 1$. This completes the base bootstrapping procedure, performing the slot-wise map

$$\frac{p^r}{p} \cdot m_i + e_i \mapsto m_i$$

. Note that the RLWE plaintext space has now been set down from p^r to p as part of this procedure.

The cost of digit extraction is dominated by $\frac{r(r-1)}{2}$ evaluations of the degree p lifting polynomial, with the degree of the procedure being p^{r-1}.

Polynomial Evaluation. From here, Lemma 3 is utilized to form

$$\tilde{f}(X) = f(0) - \sum_{i=1}^{p-1} X^i \sum_{a=0}^{p-1} f(a) a^{p-1-i}$$

as per Section 2.3. As the RLWE plaintext space is set to $p = t_{\text{lwe}}$, we can directly use $f(X)$, where $f : \mathbb{Z}_p \to \mathbb{Z}_p$ is the function which we want to evaluate on messages m_i.

RLWE to LWE Extraction. Modulus switching, key switching and RLWE to LWE extraction are then applied. Note that prior to this, the slots to coefficients map is applied, followed by a conversion from BGV to BFV encoding. Modulus

switching and key switching are then applied and directly follow from Section 2.3.

For RLWE to LWE extraction, let $(a = \sum_{i=0}^{N_{\text{rlwe}}-1} a_i X^i, b = \sum_{i=0}^{N_{\text{rlwe}}-1} b_i X^i) \in \mathcal{R}^2_{Q_{\text{rlwe}}}$ be the RLWE ciphertext obtained after modulus switching and key switching. ℓ LWE ciphertexts $(\overrightarrow{a_i}', b_i') \in \mathbb{Z}^{n_{\text{lwe}}+1}_{q_{\text{lwe}}}$ are then obtained, for $0 \leq i \leq \ell - 1$, where $a'_{i,j} = a_{i \cdot d - j}$ for $j \leq i \cdot d$, $a'_{i,j} = -a_{(\ell+i) \cdot d - j}$ for $j > i \cdot d$, and $b_i' = b_{i \cdot d}$, where $0 \leq j \leq n_{\text{lwe}} - 1$ and $0 \leq i \leq \ell - 1$.

Let RLWE $sk = \sum_{j=0}^{N_{\text{rlwe}}-1} sk_j X^j$. Note that $sk_j = 0$ for $j \geq n_{\text{lwe}}$ after key switching. The LWE secret key $\overrightarrow{sk}' \in \mathbb{Z}^{n_{\text{lwe}}}_{q_{\text{lwe}}}$ is obtained by setting $sk'_j = sk[j]$ for $0 \leq j \leq n_{\text{lwe}} - 1$.

Correctness follows directly from [31], where their procedure is described in Section 2.3. The procedure is applied here to extract ℓ many b_i', as opposed to their original N_{rlwe} many, each spaced out at intervals of d in the coefficients.

4.2 Analysis.

As discussed in Section 4.1, the digit extraction procedure is dominated by $\frac{r(r-1)}{2}$ evaluations of the degree p lifting polynomial. As covered in Section 4.1, one degree $p-1$ polynomial is evaluated to perform the homomorphic lookup table evaluation. Given that any degree n polynomial can be evaluated in $O(\sqrt{n})$ ciphertext-ciphertext multiplications using the Paterson-Stockmeyer algorithm [36], the cost of Digit Extraction and LUT evaluation is dominated by $(\frac{r(r-1)}{2} + 1) \cdot O(\sqrt{p})$ ciphertext-ciphertext multiplications. Each evaluation of the Paterson-Stockmeyer algorithm also consumes p many plaintext-ciphertext multiplications for evaluation of a degree p polynomial. There is therefore a consumption of $(\frac{r(r-1)}{2} + 1) \cdot p$ plaintext-ciphertext multiplications.

Comparison. We compare this to the work of [31], which instead only performs the lookup table evaluation. In their case however, due to the use of large primes, this was performed by evaluating a degree 65536 polynomial, requiring $O(\sqrt{t'_{\text{rlwe}}})$ ciphertext-ciphertext multiplications and t'_{rlwe} plaintext-ciphertext multiplications where $t'_{\text{rlwe}} = 65537$. By selecting values of p to be small enough and appropriate values of r, the procedures of Section 4.1 outperform theirs overall. This is experimentally verified in Section 7.

5 Improved Functional Bootstrapping

By putting together the techniques from Sections 3 and 4, we achieve lower latency functional bootstrapping.

Set LWE parameters to have plaintext space p, ciphertext space $q_{\text{lwe}} = p^r$ and dimension n_{lwe}. Set RLWE parameters to have plaintext space $t_{\text{rlwe}} = p^r$, ciphertext space Q_{rlwe}, number of slots ℓ and slot degree d. Note that the procedure is performed using BGV, therefore requiring an encoding switch via multiplication by a constant, prior to RLWE to LWE extraction.

Functional Bootstrapping Procedure

1. Linear Transformation
   - Given ℓ many LWE ciphertexts $(\overrightarrow{a_i}, b_i) \in \mathbb{Z}_q^{n_{\text{lwe}}+1}$, $0 \leq i \leq \ell - 1$ sharing the same secret key $\overrightarrow{sk} \in \mathbb{Z}^{n_{\text{lwe}}}$.
   - Perform the Laurent Polynomial Encoding for each LWE $(\overrightarrow{a_i}, b_i)$ (Sect. 3.2). This yields either $\frac{n_{\text{lwe}}}{d}$ or $\frac{n_{\text{lwe}}}{d/2}$ many RLWE plaintexts and ciphertexts, depending on the value of p (Sect. 3.4).
   - Each plaintext-ciphertext pair is then multiplied and a sum is computed over all products. A $\mathbb{Z}_{p^r}$-*linear* map π_0 is applied to obtain the inner product $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ in the slots (Sect. 3.3).
   - This result is then subtracted from a plaintext containing b_i in each slot i, obtaining $b_i - \langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle = \frac{p^r}{p} \cdot m_i + e_i$ slot-wise.
2. Digit Extraction
   - Perform digit extraction (Sect. 4.1) to apply $\frac{p^r}{p} \cdot m_i + e_i \mapsto m_i$ slot-wise.
3. Polynomial Evaluation
   - Evaluate the polynomial corresponding to the LUT (Sect. 4.1) to obtain the desired $f(m_i)$ slot-wise.
4. Slots to Coefficients
   - Move slot entries to the coefficients of the encoded polynomial to obtain an encryption of RLWE plaintext $m = \sum_{i=0}^{N_{\text{rlwe}}-1} f(m_i) X^i$ (Sect. 2.3).
5. RLWE to LWE Extraction
   - Multiply by a constant to move from BGV to BFV encoding (Sect. 2.2.2).
   - Perform the RLWE to LWE extraction (Sect. 4.1), including modulus switching and key switching, to obtain the desired LWE ciphertexts.

5.1 Analysis

5.1.1 Noise From the correctness of BGV, the BGV ciphertext after modulus switching and key switching has error $\|e\|_\infty \leq \frac{q_{\text{lwe}}}{2p}$. The error follows a Gaussian distribution centered at 0. As discussed in [8], the extraction procedure does not increase noise. The resulting ℓ LWE ciphertexts $(\overrightarrow{a_i}', b_i') \in \mathbb{Z}_{q_{\text{lwe}}}^{n_{\text{lwe}}+1}$, $0 \leq i \leq \ell - 1$, therefore each have error $\leq \frac{q_{\text{lwe}}}{2p}$.

5.1.2 Security Security follows by first selecting security parameter λ. From here, appropriate LWE parameters $n_{\text{lwe}}, q_{\text{lwe}}, \mathcal{D}_{\text{lwe}}, \chi_{\sigma_{\text{lwe}}}$ are selected, satisfying $Pr[e < \lfloor \frac{q_{\text{lwe}}}{2p} \rfloor] \geq 1 - negl(\lambda)$, $e \leftarrow \chi_{\sigma_{\text{lwe}}}$. This, together with the hardness assumption of the decisional LWE problem described in Section 2.1, guarantees the semantic security of the input LWE ciphertexts. Semantic security of the ciphertexts from Sections 3 to 4 follows from the security of the BGV keys, the appropriate RLWE parameters $N_{\text{rlwe}}, Q_{\text{rlwe}}, \mathcal{D}_{rlwe}, \chi_{\sigma_{rlwe}}$ selected and the hardness assumption of the decisional RLWE problem, described in Section 2.1 and discussed in [4].

Additionally, circular security is also assumed. This is required for the following portions of our scheme. In the Linear Transformation portion described

in Section 3, LWE secret keys are encrypted under RLWE secret keys. Furthermore, when BGV rotations are performed, the rotated RLWE secret keys are encrypted under the RLWE secret key as rotation keys. When BGV key switching is performed, the RLWE secret key is encrypted under itself too. The RLWE secret key is encrypted under the LWE secret key for the key switching procedure described in Section 4.1. The security of these components is guaranteed by the circular security assumption. The bootstrapping procedure is therefore secure.

5.1.3 Efficiency As discussed in Sections 3.5 and 4.2, the new bootstrapping framework therefore requires $\frac{n_{\text{lwe}}}{cd} + 1 + (\frac{r(r-1)}{2} + 1) \cdot p + \ell$ plaintext-ciphertext multiplications, where $c \in \{\frac{1}{2}, 1\}$, $\log d + O(\sqrt{\ell})$ rotations, and $(\frac{r(r-1)}{2}+1) \cdot O(\sqrt{p})$ ciphertext-ciphertext multiplications. It is noted that the additional ℓ terms come from evaluating slots to coefficients map.

Comparison. We compare our bootstrapping framework to the full bootstrapping procedure of [31]. An overview is provided in Table 1. Denote their slot count and RLWE plaintext prime as ℓ' and t'_{rlwe} respectively. For plaintext-ciphertext multiplications, as discussed in Section 3.5, we have that $\frac{n_{\text{lwe}}}{cd} + 1 < n_{\text{lwe}}$ by a factor of approximately cd. From our choice of parameters, $(\frac{r(r-1)}{2} + 1) \cdot p + \ell < t'_{\text{rlwe}} + \ell'$ holds. Similarly, for rotations and ciphertext-ciphertext multiplications, as discussed in Section 4.2, the usage of small primes p allows both d and ℓ to be sufficiently small. It then follows from our parameter choice, as well as Sections 3.5 and 4.2, that our framework outperforms theirs for each operation.

Operations	Laurent Polynomial-Based Bootstrapping	Bootstrapping Framework of [31]
Plaintext-Ciphertext Multiplications	$\frac{n_{\text{lwe}}}{cd} + 1 + (\frac{r(r-1)}{2} + 1) \cdot p + \ell$, $c \in \{\frac{1}{2}, 1\}$	$n_{\text{lwe}} + t'_{\text{rlwe}} + \ell'$
Rotations	$\log d + O(\sqrt{\ell})$	$O(\sqrt{n_{\text{lwe}}} + \sqrt{\ell'})$
Ciphertext-Ciphertext Multiplications	$(\frac{r(r-1)}{2} + 1) \cdot O(\sqrt{p})$	$O(\sqrt{t'_{\text{rlwe}}})$

Table 1: Comparison of efficiency for our bootstrapping framework and the bootstrapping framework of [31].

6 Multi-valued Functional Bootstrapping

Multi-valued functional bootstrapping enables the evaluation of multiple functions on a given input. To extend the functional bootstrapping procedure of Section 5 to that of a multi-valued variant, we proceed as follows. Parameters follow

from Section 5. First, Steps 1 and 2 of the functional bootstrapping procedure in Section 5 are performed once. For the polynomial evaluation portion given by Step 3 of Section 5, the computation of powers of X^i, for $1 \leq i \leq \lceil \sqrt{p-1} \rceil$ and where X is the input ciphertext of the polynomial, is performed once, as part of the Paterson-Stockmeyer algorithm [36]. This was described in Section 2.3. Note that the computation of powers of X^i can be first performed here as they are independent of the function to be evaluated. Following this, the remaining portion of Step 3, as well as Steps 4 and 5 of Section 5 are run for every individual distinct function to be evaluated.

Multi-valued Functional Bootstrapping Procedure

1. Linear Transformation
   - Given ℓ many LWE ciphertexts $(\overrightarrow{a_i}, b_i) \in \mathbb{Z}_q^{n_{\text{lwe}}+1}$, $0 \leq i \leq \ell - 1$ sharing the same secret key $\overrightarrow{sk} \in \mathbb{Z}^{n_{\text{lwe}}}$.
   - Perform the Laurent Polynomial Encoding for each LWE $(\overrightarrow{a_i}, b_i)$ (Sect. 3.2). This yields either $\frac{n_{\text{lwe}}}{d}$ or $\frac{n_{\text{lwe}}}{d/2}$ many RLWE plaintexts and ciphertexts, depending on the value of p (Sect. 3.4).
   - Each plaintext-ciphertext pair is then multiplied and a sum is computed over all products. A $\mathbb{Z}_{p^r}$-*linear* map π_0 is applied to obtain the inner product $\langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle$ in the slots (Sect. 3.3).
   - This result is then subtracted from a plaintext containing b_i in each slot i, obtaining $b_i - \langle \overrightarrow{a_i}, \overrightarrow{sk} \rangle = \frac{p^r}{p} \cdot m_i + e_i$ slot-wise.
2. Digit Extraction
   - Perform digit extraction (Sect. 4.1) to apply $\frac{p^r}{p} \cdot m_i + e_i \mapsto m_i$ slot-wise.
3. Polynomial Evaluation
   - Perform the first portion of the Paterson-Stockmeyer algorithm which computes powers of X^i, where X is the input ciphertext of the polynomial and $1 \leq i \leq \lceil \sqrt{p-1} \rceil$ (Sect. 4.1).

For each given LUT function f:

4. Polynomial Evaluation
   - Evaluate the polynomial corresponding to the LUT (Sect. 4.1) to obtain the desired $f(m_i)$ slot-wise. This is performed using the remainder of the Paterson-Stockmeyer algorithm.
5. Slots to Coefficients
   - Move slot entries to the coefficients of the encoded polynomial to obtain an encryption of RLWE plaintext $m = \sum_{i=0}^{N_{\text{rlwe}}-1} f(m_i) X^i$ (Sect. 2.3).
6. RLWE to LWE Extraction
   - Multiply by a constant to move from BGV to BFV encoding (Sect. 2.2.2).
   - Perform the RLWE to LWE extraction (Sect. 4.1), including modulus switching and key switching, to obtain the desired LWE ciphertexts.

6.1 Analysis

Analysis of the noise and security both follow directly from Section 5.

6.1.1 Efficiency Let us assume there are k distinct LUT functions to be evaluated. Steps 1, 2 of the multi-valued functional bootstrapping procedure share the same complexity as that in Section 5. Step 3 utilizes $\sqrt{p}$ ciphertext-ciphertext multiplications as described in Section 2.3. For each additional function to be evaluated, Step 4 takes p plaintext-ciphertext multiplications and $\sqrt{p}$ ciphertext-ciphertext multiplications. For each additional function to be evaluated, Steps 5 and 6 follow that of Section 5. Each additional function evaluated therefore adds extra $p+\ell$ plaintext-ciphertext multiplications, $O(\sqrt{\ell})$ rotations and $O(\sqrt{p})$ ciphertext-ciphertext multiplications.

The multi-valued functional bootstrapping framework therefore requires $\frac{n_{\mathrm{lwe}}}{cd}+1+(\frac{r(r-1)}{2}+k)\cdot p+k\cdot\ell$ plaintext-ciphertext multiplications, where $c\in\{\frac{1}{2},1\}$, $\log d+O(k\cdot\sqrt{\ell})$ rotations, and $(\frac{r(r-1)}{2}+k)\cdot O(\sqrt{p})$ ciphertext-ciphertext multiplications in total.

Operations	Laurent Polynomial-Based Multi-valued Functional Bootstrapping	Multi-valued Variant of [31]
Plaintext-Ciphertext Multiplications	$\frac{n_{\mathrm{lwe}}}{cd}+1+(\frac{r(r-1)}{2}+k)\cdot p+k\cdot\ell$, $c\in\{\frac{1}{2},1\}$	$n_{\mathrm{lwe}}+k'\cdot t'_{\mathrm{rlwe}}+k'\cdot\ell'$
Rotations	$\log d+O(k\cdot\sqrt{\ell})$	$O(\sqrt{n_{\mathrm{lwe}}}+k'\cdot\sqrt{\ell'})$
Ciphertext-Ciphertext Multiplications	$(\frac{r(r-1)}{2}+k)\cdot O(\sqrt{p})$	$k'\cdot O(\sqrt{t'_{\mathrm{rlwe}}})$

Table 2: Comparison of efficiency for our multi-valued functional bootstrapping framework and the multi-valued variant of [31].

Comparison. We compare our multi-valued functional bootstrapping framework to a multi-valued variant of the bootstrapping procedure in [31]. This variant is intuitively obtained by simply repeating each step of [31] from the DRaM polynomial evaluation step onwards for each given function. This excludes the computation of the powers of X^i in the Paterson-Stockmeyer algorithm, similar to our framework. An overview is provided in Table 2. Denote their slot count, RLWE plaintext prime and number of distinct functions to be evaluated as ℓ', t'_{rlwe} and k' respectively. As discussed in Section 3.5, for plaintext-ciphertext multiplications we have that $\frac{n_{\mathrm{lwe}}}{cd}+1<n_{\mathrm{lwe}}$ by a factor of approximately cd. By our choice of parameters, $(\frac{r(r-1)}{2}+k)\cdot p+\ell<k'\cdot t'_{\mathrm{rlwe}}+k'\cdot\ell'$ holds. Similarly, for rotations and ciphertext-ciphertext multiplications, as discussed in Section 4.2, the usage of small prime p allows both d and ℓ to be sufficiently small. Parameter k is also chosen such that the efficiency of our framework outperforms theirs for those operations. The same analysis applies when $k'=k$ or $k'=1$, with our multi-valued functional bootstrapping outperforming the multi-valued variant

of [31]. It therefore follows from the choice of parameters, as well as Sections 3.5 and 4.2, that our framework outperforms theirs for each operation.

7 Experiments

The bootstrapping frameworks presented in Sections 5 and 6 are implemented in C++ using HElib [24]. Specifically, for functional bootstrapping, we first implement the bootstrapping framework using the Laurent polynomial-based linear transformation as per Section 5. This utilized power-of-two cyclotomic m. Following this, a non-power-of-two cyclotomic m variant of our framework is implemented, utilizing repeated calls to the linear transformation as described in Section 3.1. Lastly, this is benchmarked against components of the bootstrapping framework of [31] implemented within HElib. For multi-valued bootstrapping, the scheme described in Section 6 is implemented and benchmarked against components of a multi-valued variant of [31]. The implementations were benchmarked on an Intel(R) Xeon(R) Platinum 8170 with maximum turbo frequency of 3.7 GHz and 192 GB RAM.

Parameter Set	p^r	m	N_{rlwe}	n_{lwe}	d	ℓ	Security (bits)
1	127^3	65536	32768	1024	512	64	84.8
2	257^3				256	128	78.647
3		131072	65536		512		128

Table 3: Parameters for our bootstrapping framework described in Section 5.

Functional Bootstrapping. Primes $p = 127 \approx 2^7$ and $p = 257 \approx 2^8$ were used, with $r = 3$. Similar to [31], cyclotomic $m = 65536$ and LWE dimension $n_{\text{lwe}} = 1024$ were utilized. In addition to this, cyclotomic $m = 131072$ was also utilized for the $p = 257$ case. The parameter sets used are listed in Table 3. The secret keys were sampled uniformly from a ternary distribution. Experiments of our bootstrapping framework were conducted and the results are presented in Table 4.

Additionally, for better comparison, we explore the use of non-power-of-two cyclotomic m with small plaintext prime-powers. These parameters are listed in Table 5. Crucially, we note that these parameters limit the linear transformation technique to that of Section 3.1 rather than our optimized Laurent polynomial-based one. This is due to the slot modulus $F'_i(X)$ not necessarily having a sparse shape as per what was described in Lemma 1, resulting in wrap-around when performing the convolution described in Section 3.4. The main operations, the Linear Transformation, Digit Extraction and Lookup Table evaluation, were

Param. Set	Timings(s)					Total Time(s)	Amortized Time(ms)
	LinTrans	DigitExt	LUTEval	SlotstoCoeff	RLWEtoLWE		
1	2.679	29.019	6.468	10.388	1.319	49.873	779.267
2	9.409	86.968	15.194	130.834	1.561	243.967	1905.990
3	7.492	131.590	32.204	94.855	3.236	269.376	2104.503

Table 4: Timings for our bootstrapping framework described in Section 5.

benchmarked in Table 6. The m values were kept to be large for better comparison with our parameters.

Parameter Set	p^r	m	N_{rlwe}	n_{lwe}	d	ℓ	Security (bits)
4	127^3	47749	44064	1024	18	2448	96.5
5		41401	39204			2178	83.7
6	257^3	46741	45612		6	7602	84.9

Table 5: Parameters for non-powers-of-two m.

As observed from Table 6, our framework presented in Table 4 performs significantly better in terms of latency for similar parameters. Most notably, our Laurent polynomial linear transformation of parameter set 1 outperforms that of parameter set 4 by a factor of 158.65×. Comparing the total time for the three components of those two parameter sets, our Laurent polynomial-based bootstrapping outperforms that of parameter set 4 by a factor of 12.94×. However, due to the difference in the number of slots ℓ, the amortized timings of Table 4 are worse than that of Table 6. The non-power-of-two cyclotomic m variant therefore offers an alternative framework that prioritizes a more balanced amortized time at the cost of increased latency compared to our framework in Section 5.

Param. Set	Timings(s)			Total Time(s)	Amortized Time(ms)
	LinTrans	DigitExt	LUTEval		
4	425.02	54.426	14.375	493.820	201.724
5	208.367	53.088	13.812	275.268	126.386
6	300.839	100.579	26.707	428.126	56.317

Table 6: Timings for non-powers-of-two m bootstrapping.

Lastly, for comparison, the Homomorphic Linear Transformation and the LUT Polynomial Evaluation components of [31] were also implemented in HElib. The benchmarks are presented in Table 7. As observed, a significant improvement of at least 26.95× in latency is obtained when comparing with the $p^r = 127^3$ case of our framework in Table 4. For the $p^r = 257^3$ case, an improvement in latency of 5.51× and 4.99× is obtained.

Component	t'_{rlwe}	p^r	m	N_{rlwe}	n_{lwe}	d	ℓ'	Security (bits)	Total Time(s)
Homomorphic Linear Transformation	65537	2^9	65536	32768	1024	1	32768	80.9551	314.257
LUT Polynomial Evaluation									1029.974

Table 7: Timings for the bootstrapping framework of [31].

However, we note that compared to [31], the amortized timings of our framework were impacted, experiencing a slowdown of at most 19×, 46.46× and 51.30× respectively for each parameter set. This likely arises due to the difference in the number of slots, ℓ and ℓ', which is significantly larger in the bootstrapping framework of [31]. Furthermore, it is also noted that in the case of $p^r = 257^3$ with $m = 65536$, performance was likely affected by increasing the number of columns in the key-switching matrix to 30 within the HElib implementation, in order to achieve sufficient security. Compared to the non-power-of-two cyclotomic m case, it is observed that for latency [31] faces up to 4.88× slowdown in total time for these components. However, for these components, up to 4.92× improvement in amortized time is obtained for [31].

Component	Laurent Poly. Timings(s)						Timings of framework of [31](s)
	p^r	N_{rlwe}	p^r	N_{rlwe}	p^r	N_{rlwe}	
	127^3	32768	257^3	32768	257^3	65536	
Linear Transformation	2.679		9.409		7.492		314.257
Digit Extraction	35.487		102.162		163.794		-
LUT Evaluation							1029.974

Table 8: Comparison of component timings for our bootstrapping framework described in Section 5 and the bootstrapping framework of [31].

A component-wise comparison with our framework of Section 5 is further illustrated in Table 8. A comparison of the Linear Transformation, Digit Extraction and LUT Evaluation components is performed as those form the main

differences between the two bootstrapping frameworks. As observed from Table 8, our Laurent polynomial-based linear transformation is significantly faster than theirs in terms of latency, with latency improvements of 117.3×, 33.4× and 41.9× respectively for that component. Furthermore, while our framework requires both Digit Extraction and LUT Evaluation as opposed to just LUT Evaluation in [31], when put together the combined components still outperform the single LUT Evaluation of theirs. Specifically a latency improvement of 29.02×, 10.08× and 6.29× is obtained for that component.

The three functional bootstrapping frameworks presented here can thus be summarized as follows. Our bootstrapping procedure of Section 5, utilizing the power-of-two cyclotomic m presented in Tables 3 and 4, represents a variant that provides low latency at the expense of amortized time. The non-power-of-two m case, presented in Tables 5 and 6, represents a variant that balances latency with amortized runtime. Finally, the bootstrapping framework of [31], presented in Table 7, represents a variant that minimizes amortized time at the expense of latency.

Multi-valued Functional Bootstrapping. For the multi-valued functional bootstrapping of Section 6, a breakdown of the timings is provided in Table 9. Experiments were run for parameter sets 1, 3 and the benchmark parameters of Table 7. Here, the multi-valued precomputation corresponds to Steps 1 to 3 of the multi-valued bootstrapping procedure in Section 6. The per-function extra processing time corresponds to Steps 4 to 6 of the multi-valued bootstrapping procedure in Section 6. For the benchmark parameters of Table 7, the multi-valued precomputation refers to the homomorphic linear transformation and the computation of the powers of X^i in [31]. The per-function extra processing time here refers to the LUT Evaluation step in [31]. This excludes the computation of the X^i powers, as described in Section 6.

Param. Set	Multi-valued Precomputation(s)	Per-Function Extra Processing Time(s)
1	34.458	15.415
3	147.280	122.097
Table 7	441.807	902.424

Table 9: Comparison of timings for multi-valued functional bootstrapping.

As observed in Table 9, our framework outperforms theirs for both the multi-valued precomputation and the per-function extra processing time components. The multi-valued precomputation of parameter sets 1 and 3 obtains a 12.82× and 3× improvement respectively over the benchmark parameters of Table 7. For each additional function processed, parameters sets 1 and 3 obtain at least a 58.54× and 7.39× improvement over the benchmark parameters of Table 7.

In the case where a single function is evaluated using the framework of [31], our proposed framework allows for the evaluation of at least 84 distinct functions in the same amount of time using parameter set 1.

8 Conclusion

In conclusion, we extended the bootstrapping framework proposed by [31] to that of the small primes case. Furthermore, we proposed the Laurent polynomial-based linear transformation which improved upon the homomorphic linear transformation in their framework. Put together, all these resulted in an improvement of at least 26.95× in latency compared to the original. Additionally, this lower latency enabled a new multi-valued functional bootstrapping framework, which allowed the evaluation of at least 84 distinct functions in the same amount of time required for performing one round of functional bootstrapping in [31].

References

1. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 1–20. Springer, Heidelberg (Aug 2013)
2. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology 14(1), 316–338 (2020), https://doi.org/10.1515/jmc-2019-0026
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (Aug 2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012)
5. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 106–126. Springer, Heidelberg (Mar 2019)
6. Castryck, W., Iliashenko, I., Vercauteren, F.: Homomorphic SIM^{2}D operations: Single instruction much more data. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 338–359. Springer, Heidelberg (Apr / May 2018)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Heidelberg (Dec 2017)
8. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (Dec 2016)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology 33(1), 34–91 (Jan 2020)
10. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. pp. 670–699. LNCS, Springer, Heidelberg (2021)

11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
12. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816 (2014), https://eprint.iacr.org/2014/816
13. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (Apr 2015)
14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
15. Geelen, R.: Bootstrapping Algorithms for BGV and FV. Master's thesis, KU Leuven (2021)
16. Geelen, R.: Revisiting the slot-to-coefficient transformation for bgv and bfv. Cryptology ePrint Archive (2024)
17. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. pp. 257–286. LNCS, Springer, Heidelberg (2023)
18. Geelen, R., Vercauteren, F.: Bootstrapping for BGV and BFV revisited. Journal of Cryptology 36(12), 1432–1378 (Mar 2023)
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009)
20. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR TCHES 2021(2), 229–253 (2021), https://tches.iacr.org/index.php/TCHES/article/view/8793
21. Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (Aug 2014)
22. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (Apr 2015)
23. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 93–120. Springer, Heidelberg (Aug 2018)
24. IBM: HElib. https://github.com/homenc/HElib/ (2023)
25. Janneck, J., Tueno, A., Kußmaul, J., Akram, M.: Private computation on set intersection with sublinear communication. Cryptology ePrint Archive, Report 2022/1137 (2022), https://eprint.iacr.org/2022/1137
26. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 1651–1669. USENIX Association (Aug 2018)
27. Kim, J., Seo, J., Song, Y.: Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from ckks. Cryptology ePrint Archive (2024)
28. Kluczniak, K., Schild, L.: Fdfb ^{2}: Functional bootstrapping via sparse polynomial multiplication. Cryptology ePrint Archive (2024)
29. Lee, D., Min, S., Song, Y.: Functional bootstrapping for packed ciphertexts via homomorphic lut evaluation. Cryptology ePrint Archive (2024)
30. Li, Z., Shen, X., Lu, X., Wang, R., Zhao, Y., Wang, Z., Wei, B.: Leveled functional bootstrapping via external product tree. Cryptology ePrint Archive (2025)

31. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7 ms, with $\tilde{O}(1)$ polynomial multiplications. pp. 101–132. LNCS, Springer, Heidelberg (2023)
32. Liu, Z., Wang, Y.: Relaxed functional bootstrapping: A new perspective on bgv/bfv bootstrapping. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 208–240. Springer (2025)
33. jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. pp. 1057–1073. IEEE Computer Society Press (2021)
34. Meftah, S., Tan, B.H.M., Mun, C.F., Aung, K.M.M., Veeravalli, B., Chandrasekhar, V.: Doren: toward efficient deep convolutional neural networks with fully homomorphic encryption. IEEE Transactions on Information Forensics and Security 16, 3740–3752 (2021)
35. Mono, J., Kluczniak, K., Güneysu, T.: Improved circuit synthesis with multi-value bootstrapping for fhew-like schemes. IACR Transactions on Cryptographic Hardware and Embedded Systems 2024(4), 633–656 (2024)
36. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing 2(1), 60–66 (1973)
37. Roman, S.: Field theory, vol. 158. Springer Science & Business Media (2005)
38. Sim, J.J., Chan, F.M., Chen, S., Meng Tan, B.H., Mi Aung, K.M.: Achieving gwas with homomorphic encryption. BMC medical genomics 13, 1–12 (2020)
39. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations 71(1), 57–81 (2014)