HasteBoots: Proving FHE Bootstrapping in Seconds

Fengrun Liu[1,3], Haofei Liang[2], Tianyu Zhang[2], Yuncong Hu[*,2], Xiang Xie[3], Haisheng Tan[1], Yu Yu[2]

[1]*University of Science and Technology of China,* [2]*Shanghai Jiao Tong University,* [3]*Primus*

Abstract—**Fully Homomorphic Encryption (FHE) enables computations on encrypted data, ensuring privacy for outsourced computation. However, verifying the integrity of FHE computations remains a significant challenge, especially for bootstrapping, the most computationally intensive operation in FHE. Prior approaches, including zkVM-based solutions and general-purpose SNARKs, suffer from inefficiencies, with proof generation times ranging from several hours to days. In this work, we propose HasteBoots, a succinct argument tailored for FHE operations. By designing customized polynomial interactive oracle proofs and optimized polynomial commitment schemes, HasteBoots achieves proof generation in a few seconds for FHE bootstrapping, significantly outperforming existing methods. Our approach demonstrates the potential for scalable and efficient verifiable FHE, paving the way for practical, privacy-preserving computations.**

1. Introduction

Fully Homomorphic Encryption (FHE) [1, 2, 3, 4, 5, 6, 7] is a groundbreaking cryptographic technology that allows computations to be performed directly on encrypted data. This unique capability ensures that sensitive information remains protected throughout the computation process, making FHE especially valuable in scenarios that require privacy preservation, such as secure cloud computing and confidential data processing. However, achieving full homomorphism—where any arbitrary computation can be performed on encrypted data—relies on a key process known as bootstrapping [1]. Bootstrapping is essential in FHE because it periodically reduces accumulated noise in the ciphertext, enabling ongoing computation without data degradation, thereby maintaining both security and correctness.

In secure outsourced computing, Fully Homomorphic Encryption is often deployed to protect client data privacy, as it allows computations to be carried out on encrypted data. However, this setting introduces a unique challenge: while FHE ensures data confidentiality, it does not inherently guarantee that the computations performed on encrypted data are accurate or unaltered. For example, in outsourced machine learning scenarios—such as neural network models handled by cloud service providers—the client must have assurance that the inference or training computations on their model are correct, despite not being able to access the raw (unencrypted) data or independently verify the results.

$*$ Corresponding author

This is where Succinct Non-Interactive Arguments of Knowledge (SNARKs) [8, 9, 10, 11, 12, 13, 14, 15, 16] become essential. SNARKs provide a way to generate a compact, verifiable proof that the computations were executed correctly. This proof, which can be verified efficiently, allows the client to check the correctness of the computation without needing to trust the service provider. In decentralized systems, where the integrity of computations is crucial, FHE combined with SNARKs addresses the trust issue by providing verifiable results.

While some prior work [17, 18, 19, 20, 21, 22, 23] managed to prove basic FHE operations, such as addition and multiplication, these methods fail to support complex FHE operations, particularly bootstrapping. Bootstrapping is essential for achieving full homomorphism, as it resets accumulated noise in ciphertexts to allow ongoing computations. However, it is also the most computationally intensive operation in FHE, and how to prove bootstrapping efficiently remains a significant open problem in verifiable FHE.

One attempt is to leverage zero-knowledge virtual machines (zkVMs) [24, 25], which implement FHE algorithms directly in high-level language such as Rust, and then compile it to a proof. Although feasible, these zkVM-based solutions suffer from extremely low efficiency. Compiled code is not optimized for SNARKs, resulting in prohibitively long proof generation time, particularly for bootstrappings. Consequently, existing zkVMs like RISC0 [24] and SP1 [25] require approximately **three days** and **one day**, respectively, to generate a proof for a single bootstrapping.

A recent work [26] uses general-purpose SNARKs to directly verify FHE operations. [26] obtains performance improvements by eliminating the compilation from high-level languages. However, general-purpose SNARKs are not inherently optimized for FHE. For instance, FHE operations often involve complex relations such as rounding, Number Theoretic Transform (NTT), and transformation from a ring element to a polynomial of a multiplicative group. These specialized relations cannot be efficiently presented in the general circuits or custom lookup gates supported by existing general-purpose SNARK frameworks, leading to efficiency loss. As a result, [26] needs about **half an hour** for proving bootstrapping. This long proof generation time limits the practical scalability of verifiable FHE.

Our goal is to construct a proof system specifically tailored for FHE that can generate proofs in just **a few seconds**, making verifiable FHE computations viable for scalable, privacy-preserving computations without centralized trust.

1.1. Our Contributions

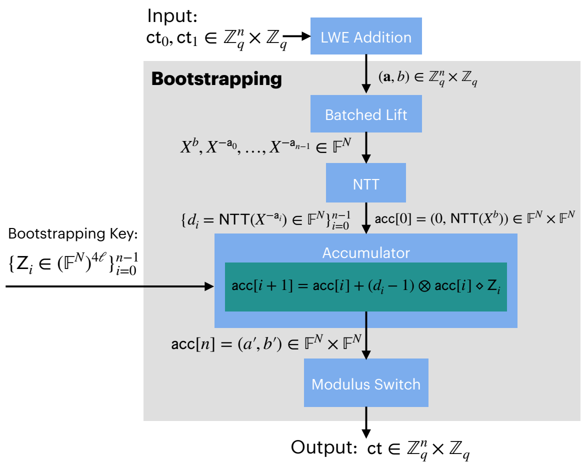
Our approach leverages the Polynomial Interactive Oracle Proofs (PIOPs) and Polynomial Commitment Schemes (PCS) paradigm to construct an efficient proof system specifically for FHE. Our contributions are as follows:

- **We propose a novel proof construction framework for FHE**. We identify key atomic operations in FHE workflows as shown in Fig. 1 and build a new framework for proof construction as shown in Fig. 2. We observe that the FHE workflow does not involve private input so that knowledge soundness and zero-knowledge are not required in our proof system. Based on this framework, we analyze FHE algorithms and identify the FHEW/TFHE [6, 7] as the *proof-friendly FHE* scheme due to its simpler bootstrapping process, which is the most expensive operation in FHE. We further select FHE parameters that enable more efficient proof generations. Our framework provides a foundation for future work to optimize proofs for FHE where it suffices to improve the sub-protocols for each component of the framework.
- **We construct an efficient succinct argument HasteBoots for FHEW/TFHE**. We develop custom PIOPs tailored for FHE operations. These PIOPs leverage the specific structures of these operations to significantly enhance proof efficiency. Notably, we achieve an optimal prover for fast-NTT algorithm with bit-reversal, which can be of independent interest. Additionally, our approach supports batch proof of multiple operations, further reducing computational overhead. Considering proving efficiency and the need for field-agnostic support across different FHE fields, we select a PCS based on linear codes [16]. However, we observe that FHE involves many small polynomials, which are inefficient to open with linear-code PCS. To address this, we design a packing mechanism that combines multiple small polynomials into a single large polynomial for opening. As a result, we obtain a succinct argument HasteBoots for FHEW/TFHE with linear proving time.
- **We implement and evaluate HasteBoots**. Our evaluation demonstrates the efficiency of HasteBoots, with much faster proof generation than existing general-purpose proof systems. On Apple M4, HasteBoots can prove a FHE NAND operation with bootstrapping in **3 seconds**, whereas the state-of-the-art [26] require about half an hour.

1.2. Overview of Our Techniques

Workflow of FHE NAND. We first introduce the workflow of our FHE NAND procedure in Fig. 1. It includes four main steps: LWE addition, batched lift followed by NTT operations, accumulator updating, and modulus switching.

The input consists of two $\mathbb{LWE}$ ciphertexts $\mathsf{ct}_0 = (\boldsymbol{a}_0, b_0) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ and $\mathsf{ct}_1 = (\boldsymbol{a}_1, b_1) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, encrypted under the secret key $\boldsymbol{s} \in \mathbb{Z}_q^n$. The output is a $\mathbb{LWE}$ ciphertext $\mathsf{ct} = (-\boldsymbol{a}_0 - \boldsymbol{a}_1, \frac{5q}{8} - b_0 - b_1) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$,



Input:
$\mathsf{ct}_0, \mathsf{ct}_1 \in \mathbb{Z}_q^n \times \mathbb{Z}_q$
LWE Addition
Bootstrapping
$(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$
Batched Lift
$X^b, X^{-a_0}, \dots, X^{-a_{n-1}} \in \mathbb{F}^N$
NTT
$\{d_i = \mathsf{NTT}(X^{-a_i}) \in \mathbb{F}^N\}_{i=0}^{n-1}$
$\mathsf{acc}[0] = (0, \mathsf{NTT}(X^b)) \in \mathbb{F}^N \times \mathbb{F}^N$
Bootstrapping Key:
$\{\mathsf{Z}_i \in (\mathbb{F}^N)^{4\ell}\}_{i=0}^{n-1}$
Accumulator
$\mathsf{acc}[i+1] = \mathsf{acc}[i] + (d_i - 1) \otimes \mathsf{acc}[i] \diamond \mathsf{Z}_i$
$\mathsf{acc}[n] = (a', b') \in \mathbb{F}^N \times \mathbb{F}^N$
Modulus Switch
Output: $\mathsf{ct} \in \mathbb{Z}_q^n \times \mathbb{Z}_q$

Figure 1: The computation flow of the NAND operation in FHE. The gray part represents the bootstrapping, where the bootstrapping keys are public. We assume $q = 2N$ for simplicity. $\otimes$ and $\diamond$ are defined in Section 2.2.

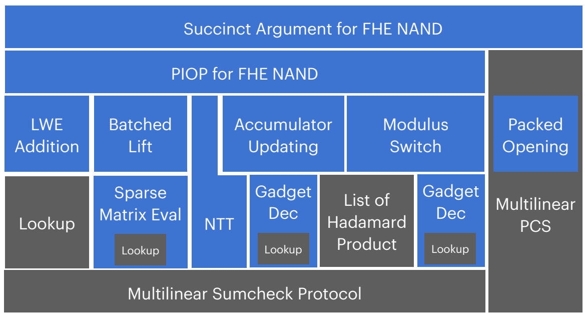


Figure 2: Protocol design overview of proof systems for FHE NAND, highlighting our newly designed protocols in blue and existing protocols in gray. Each protocol is built on top of the underlying protocols.

encrypted under the same secret key $\boldsymbol{s}$, while the corresponding plaintext is the NAND result of the input plaintext.

FHE scheme first performs LWE addition to these ciphertexts to produce $(\boldsymbol{a}, b) = (\boldsymbol{a}_0 + \boldsymbol{a}_1, b_0 + b_1)$ with increased noise, where $\frac{5q}{8}$ is omitted for simplicity (the full FHE protocol refers to Section 2.2). Bootstrapping aims to reduce this noise by homomorphically decrypting the ciphertext using the secret key encrypted under the secret key (a so-called "circular encryption"), denoted as Z_i, an $\mathbb{RGSW}$ ciphertext encrypting the secret s_i. The decryption circuit of the LWE ciphertext $(\boldsymbol{a}, b)$ is computed by $\lfloor \frac{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle}{q/4} \rceil$.

Our bootstrapping procedure uses an RLWE scheme [27] over a cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, with its quotient ring $\mathcal{R}_Q = \mathcal{R}/(Q\mathcal{R}) = \mathbb{F}[X]/(X^N + 1)$, where $\mathbb{F}$ is the finite field over the prime Q. The cyclotomic ring contains a cyclic subgroup $G = \{X, \dots, X^{2N}\}$, and a $2N$-th root of unity ω such that $\omega^{2N} = 1$. To perform the homomorphic decryption in the RLWE scheme, we lift the operation into exponent form, computing $X^{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle}$. Assuming $q = 2N$ for simplicity, the lift operation maps $b \in \mathbb{Z}_q$ to the cyclic group, resulting in a lifted polynomial $X^b \in G$. The accumulator is updated iteratively to obtain an $\mathbb{RLWE}$ ciphertext encrypting $X^{b-\langle \boldsymbol{a},\boldsymbol{s}\rangle}$ with reduced noise. Finally, we extract an $\mathbb{LWE}$ ciphertext $(\boldsymbol{a}', b')$ under modulus Q from the accumulator's output and perform modulus switching to return it to modulus q. For each component $a'_i \in \mathbb{F}$, modulus switching computes $\lfloor \frac{a'_i \cdot q}{Q} \rceil \in \mathbb{Z}_q$ by rounding to the nearest integer.

PIOP Design. We introduce our PIOP design as shown in Fig. 2 tailored for the FHE NAND workflow. We use the multilinear polynomials to encode the FHE ciphertexts. By reducing all relations to a multilinear sumcheck protocol, we achieve an efficient linear-time prover. We select the FHE modulus Q to ensure that our PIOP can be defined on the finite field $\mathbb{F}$ over the prime Q, enabling native FHE arithmetic on $\mathbb{Z}_Q$. We further employ an extension field $\mathbb{EF}$ to improve the soundness.

LWE Addition. The core of proving addition between two $\mathbb{LWE}$ ciphertexts involves an addition modulo q. However, since our proof system uses $\mathbb{F}$ with modulus Q, we need to simulate this modulus operation within the field. Given the multilinear extensions $\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$, the relation $\tilde{\boldsymbol{c}}(x) = \tilde{\boldsymbol{a}}(x) + \tilde{\boldsymbol{b}}(x) \mod q$ holds for all $x \in \{0,1\}^{\log n}$ if and only if there exists $\tilde{\boldsymbol{k}}$ such that $\tilde{\boldsymbol{a}}(x) + \tilde{\boldsymbol{b}}(x) = \tilde{\boldsymbol{k}}(x) \cdot q + \tilde{\boldsymbol{c}}(x)$ and $\tilde{\boldsymbol{k}}(x) \in \{0,1\}$. This can be done efficiently using the multilinear sumcheck protocol. We also need to prove that $\tilde{\boldsymbol{c}}(x)$ is within the correct range. We observe that since the modulus q in FHE is relatively small, it is efficient to employ a lookup argument with the table $(0, \ldots, q-1)$.

Batched Lift. Before updating the accumulator, FHE needs to perform a key operation: *lifting* the ring element $a \in \mathbb{Z}_q$ to the exponent, resulting in a polynomial X^a in the cyclic subgroup G (The exponent should be $a \cdot \frac{2N}{q}$, for simplicity, we assume $q = 2N$). Note that after lifting, the exponent a will be in $\mathbb{Z}_{2N}$. Thus, the lifting operation involves a modular operation on X^a. The modular result is a monomial $(1-2k) \cdot X^r \in \mathcal{R}_Q$, where $k = 1$ and $r = a - N$ if $a \geq N$, and $k = 0$ and $r = a$ otherwise. This monomial is represented by a coefficient vector $\boldsymbol{c} \in \mathbb{F}^N$ with a single non-zero value k, located at position r. When applying the batched lift operation to an input vector $\boldsymbol{a} = (a_0, \ldots, a_{M-1}) \in \mathbb{Z}_q^M$, we obtain a sparse coefficient matrix $C = (\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1}) \in \mathbb{F}^{N \times M}$ with only M non-zero entries, where each column $\boldsymbol{c}_i$ represents the lifted polynomial $X^{a_i} \equiv (1-2k_i) \cdot X^{r_i} \in \mathcal{R}_Q$.

A key observation is that each lifted polynomial is transformed into its evaluation form using NTT. The subclaims of proving these NTT are evaluation queries on the coefficient vectors $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1}$ at the random point. Hence, instead of explicitly proving the lift relation, we find a way to answer the queries using the input vector $\boldsymbol{a}$.

Specifically, we find that the random linear combination of all evaluations on the coefficient vectors is equal to a single evaluation of the sparse coefficient matrix $C \in \mathbb{F}^{N \times M}$ at a random point. By Schwartz-Zippel lemma, it suffices to check the random linear combination instead of individual evaluations. We employ the idea from prior work [10, 28, 29] to represent the sparse matrix by three vectors–row, col and val–that locates the non-zero entries. We can efficiently reduce the evaluation of the sparse matrix to queries on $\mathsf{row} = (r_0, \ldots, r_{M-1})$ and $\mathsf{val} = (1-2k_0, \ldots, 1-2k_{M-1})$, where $a_i = k_i \cdot N + r_i$. Note that unlike the standard solution, we can omit col since each column contains a single non-zero element. We further replace memory-checking in prior work [28][29] with an indexed lookup argument since the table size is small in the setting of the RLWE scheme.

We discuss the proof of NTT relations in the next part, as the accumulator updating process also relies on NTT.

Accumulator Updating. The accumulator initially stores an $\mathbb{RLWE}$ ciphertext $\mathsf{acc}[0]$. Each update takes a circular encryption Z_i and the lifted polynomial (after NTT) $\boldsymbol{d}_i = \mathsf{NTT}(X^{-a_i})$: $\mathsf{acc}[i+1] = \mathsf{acc}[i] + (\boldsymbol{d}_i - 1) \otimes \mathsf{acc}[i] \diamond \mathsf{Z}_i$. Both $\otimes$ and $\diamond$ (external product) are defined in Section 2.2. This process primarily involves NTT operations, Hadamard products, and gadget decompositions. The Hadamard product can be directly verified using the sumcheck protocol. We need to show how to prove NTT and gadget decompositions.

The gadget decomposition is an extension of bit decomposition that decomposes the coefficients $\boldsymbol{f} \in \mathbb{F}^N$ of a polynomial into ℓ small polynomials with coefficient vectors $\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{\ell-1} \in \mathbb{Z}_B^N$ such that $\boldsymbol{f} = \sum_{i=0}^{\ell-1} B^i \cdot \boldsymbol{f}_i$. This implies that we need to verify the relationship between $\boldsymbol{f}$ and $\boldsymbol{f}_i$ and also ensure that each $\boldsymbol{f}_i$ lies within a valid range. The former can be addressed using a polynomial identity test. For the latter, while in bit decomposition we can easily verify $(\boldsymbol{f}_i(x) - 1)\boldsymbol{f}_i(x) = 0$, gadget decomposition introduces a larger basis B, making this method inefficient. We employ a lookup protocol to verify that $\boldsymbol{f}_i(x)$ values fall within the range $[0, B)$ to further improve the performance.

The NTT operation, defined over $\mathcal{R}_Q$ with a $2N$-th root of unity ω, transforms a polynomial's coefficient vector into an evaluation vector at N points, with a Fourier matrix $F(\mathcal{Y}, \mathcal{X}) = \omega^{(2\mathcal{Y}+1)\cdot \mathcal{X}}$. Prior work [30] provides a linear-time argument for FFT with a different root of unity. However, we observe that prior methods for FFT are not directly applicable to NTT due to the different root-of-unities, which change the structure of the Fourier matrix. Additionally, optimization techniques for NTT computations (fast NTT [31]) in FHE further change the matrix structure, making it incompatible with existing approaches.

We extend prior work [30] to support a different set of roots of unities. We find that in the NTT Fourier matrix, each entry is computed based on a structured exponentiation pattern. By decomposing the exponents into bitwise components, we break down the computation into multiple rounds, which allows us to efficiently evaluate the NTT Fourier matrix using dynamic programming. To accommodate the bit-reversed order introduced by the fast NTT, we also use different decomposition orders for variables $\mathcal{X}$ and $\mathcal{Y}$.

We also propose a new method for proving batched NTT after the lift operations, which is not considered in prior work [30]. An observation is that NTT instances can be randomized into a single NTT instance due to the linearity. However, this does not save the prover time, as the prover still needs to $O(MN)$ time to compute the proof, where M

is the number of NTT instances, and N is the degree of the polynomial. We propose to leverage the sparsity of the coefficient vectors so that the prover can explicitly compute the randomized coefficient vector in linear time, followed by generating the NTT arguments for this randomized NTT instance in total complexity of $O(M+N)$ rather than $O(MN)$. This optimization is of independent interest for proving batched NTT operations on sparse polynomials.

Modulus Switching. The last step in the bootstrapping is modulus switching, which converts an $\mathbb{LWE}$ ciphertext under the modulus Q to a new $\mathbb{LWE}$ ciphertext under a smaller modulus q. Specifically, this step involves rounding the result of dividing $a \cdot q$ by Q to the nearest integer in $\mathbb{Z}_q$, as follows: $b = \lfloor \frac{a \cdot q}{Q} \rceil \mod q$. This operation maps a range of values to the same integer in $\mathbb{Z}_q$. Assuming that $2q \mid Q-1$, we define $k = \frac{Q-1}{2q}$ with the relation $Q = 2kq+1$.

We find that this relation implies that the field can be exactly divided into q ranges of length $2k$, with a single special point. As shown in Fig. 3, for any $a \in [(2b-1) \cdot k+1, (2b+1) \cdot k]$, it is mapped to $b \in \{1, \dots, q-1\}$. The remaining uncovered range, $a \in [0,k] \cup (0,-k] \equiv Q$ is mapped to 0. Although this range is composed of two separate ranges, they can be combined into a single sequential range $[(2q-1) \cdot k+1, (2q+1) \cdot k]$ of length $2k$ under modulus Q, along with a single point $a = k$. To handle this, we introduce an auxiliary witness $b' \in \{1, \dots, q\}$ such that $b' \equiv b \mod q$. This leads to a dichotomy: either $a = k$ and $b = 0$, or for any $a \in [(2b'-1) \cdot k+1, (2b'+1) \cdot k]$, a is mapped to b where $b \in \mathbb{Z}_q$, and $b' \in \{1, \dots, q\}$ with $b' \equiv b$. Finally, we leverage Hadamard products and range proofs to prove these relations. In particular, the range proofs can be constructed from our prior gadget decomposition protocols.

Polynomial Commitments. Finally, to commit multilinear polynomials, we employ polynomial commitments suited to FHE operations. We select Brakedown [32] due to its compatibility with the FHE field. Additionally, Brakedown is a linear-time protocol, which is crucial for our application.

We identify a challenge with using linear code-based PCS like Brakedown in FHE, where multiple small polynomials need to be opened. Although the total input size remains the same, Brakedown's performance drops significantly when handling multiple small polynomials compared to a single large polynomial, since the opening proofs size relies on the relative code distance. Furthermore, Brakedown lacks homomorphic properties in opening proofs, preventing batch openings. To solve this, we design a packing mechanism that combines M multilinear polynomials, each with $\log N$ variables, into a single $(\log M + \log N)$-variate polynomial. Opening this packed polynomial at a random point reveals a random linear combination of the small polynomials at a random point, reducing the overall cost.

1.3. Comparison with the state-of-the-art [26]

In [26], Thibault and Walter design a SNARK-friendly arithmetic circuit for TFHE bootstrapping procedure and prove it within plonky2, which is one feasible approach to proving custom FHE operations. However, an alternative approach is to directly prove the relation between the inputs and the outputs within these FHE operations, enabling us to achieve more efficient prover than in [26]. Extracting the relation from the arithmetic circuit and generating proofs for it is a key feature of many highly efficient protocols. For example, [33] proposed a highly optimized protocol for matrix multiplication, where the prover only needs $O(N^2)$ extra work to prove the correctness. One main bottleneck in [26] lies in the NTT operation, where they directly prove an NTT circuit with a size of $O(N \log N)$, resulting in a strictly quasi-linear prover in plonky2. In contrast, our approach achieves an optimal linear prover for the corresponding NTT relation with bit-reversal. Additionally, [26] uses a decomposition method to compute multiplication by the monomial X^a, which also leads to a circuit of size $O(N \log N)$. In our work, however, we prove this operation within our lift operation, also resulting in an optimal linear proving time. Another difference can also be observed in how we handle modulus switching. While [26] applies a decomposition approach to manage the division and rounding operations in the circuit, which is feasible but introduces additional error. Instead, we actually transform the modulus switching relation into the range check relation, which is simpler and avoids introducing extra errors.

2. Preliminary

2.1. Notation

We denote the set $\{0, \dots, a-1\}$ by $[a]$. Let $\mathbb{Z}_q$ be a ring over an integer q, and $\mathbb{F}_Q$ a field defined over a prime Q (omitted where clear). For a vector $\boldsymbol{a} \in \mathbb{F}^N$, the i-th entry is dentoed by a_i. We can decompose the vector $\boldsymbol{a}$ with respect to a base $B \in \mathbb{Z}$ into $\ell = \lceil \log_B Q \rceil$ vectors $\boldsymbol{a}_0, \dots, \boldsymbol{a}_{\ell-1}$ such that $\boldsymbol{a} = \sum_{i=0}^{\ell-1} B^i \cdot \boldsymbol{a}_i$ where $\boldsymbol{a}_0, \dots, \boldsymbol{a}_{\ell-1} \in \mathbb{Z}_B^N$. This decomposition is referred to as a gadget decomposition and is denoted by $\mathsf{Dec}(\boldsymbol{a}; B, \ell) = (\boldsymbol{a}_0, \dots, \boldsymbol{a}_\ell)$. For vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, we define the Hadamard product as $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$, where $c_i = a_i \cdot b_i$ for each i. The identity function $\mathbb{1}(x)$ returns 1 if the condition x is true, and 0 otherwise.

Bit-Reversed Representation. Throughout the paper, assume N is a power of two. Let $x \in \{0,1\}^{\log N}$ denote binary variables and $X \in \mathbb{F}$ denote a field variable. For $x = (x_0, \dots, x_{\log N-1})$, define the canonical injection $\mathsf{to\text{-}field}(x) = \sum_{i=0}^{\log N-1} 2^i \cdot x_i$, which maps $x \in \{0,1\}^{\log N}$ to $\mathbb{F}$, and let $\mathsf{to\text{-}bits}(X)$ denote its inverse. For binary strings $x, y \in \{0,1\}^{\log N}$, let $\mathcal{X}, \mathcal{Y} \in \mathbb{F}$ represent their corresponding field values for simplicity. For $\mathcal{X} \in \mathbb{F}$ corresponding to $x \in \{0,1\}^{\log N}$, define its bit-reversed representation as $\mathcal{X}^{\mathsf{R}} = \sum_{i=0}^{\log N-1} 2^{\log N-1-i} \cdot x_i$. Additionally, we denote the bit-reversed order of the vector $\boldsymbol{a}$ by $\boldsymbol{a}^{\mathsf{R}} = (a_{0^{\mathsf{R}}}, \dots, a_{(N-1)^{\mathsf{R}}}) \in \mathbb{F}^N$.

Multilinear Extensions of Vectors and Matrices. Let the identity function $eq : \{0,1\}^{\log N} \times \{0,1\}^{\log N} \to \{0,1\}$ be defined as $eq(x,y) = 1$ if $x = y$, and 0 otherwise. The multilinear extension of eq, denoted $\tilde{eq}(x,y)$, can be expressed as $\tilde{eq}(x,y) = \prod_{i=1}^{\log N} ((1-x_i)(1-y_i) + x_i y_i)$ where

$x, y \in \mathbb{F}^{\log N}$. A vector $\boldsymbol{a} = (a_0, \ldots, a_{N-1}) \in \mathbb{F}^N$ can be viewed as a multivariate polynomial $\boldsymbol{a} : \{0,1\}^{\log N} \to \mathbb{F}$ such that for all $x \in \{0,1\}^{\log N}$, we have $\boldsymbol{a}(x) = a_{\mathcal{X}}$. The multilinear extension of $\tilde{\boldsymbol{a}} : \mathbb{F}^{\log N} \to \mathbb{F}$ can be uniquely defined using the equality function as $\tilde{\boldsymbol{a}}(x_0, \ldots, x_{\log N-1}) = \sum_{b \in \{0,1\}^{\log N}} \tilde{eq}(x, b) \cdot \boldsymbol{a}(b)$ such that $\tilde{\boldsymbol{a}}(x) = \boldsymbol{a}(x)$ for all $x \in \{0,1\}^{\log N}$. Similarly, the multilinear extension of an $N \times M$ matrix A, denoted by $\tilde{A}$, is the multilinear extension of the function $A : \{0,1\}^{\log N + \log M} \to \mathbb{F}$.

Cyclotomic Ring. For N a power of two, the $2N$-th cyclotomic ring is defined as $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, which has a $2N$-th roots of unity such that $\omega^{2N} = 1$. The quotient ring $\mathcal{R}_Q = \mathcal{R}/(Q\mathcal{R})$ consists of polynomials in $\mathcal{R}$ with the coefficients modulo Q. Any polynomial $\boldsymbol{c}(X) = \sum_{i=0}^{N-1} c_i \cdot X^i \in \mathcal{R}_Q$ can be uniquely defined with the coefficient vector $\boldsymbol{c} = (c_0, \ldots, c_{N-1}) \in \mathbb{F}^N$.

NTT/INTT. The Number Theoretic Transform (NTT) enables quasi-linear polynomial multiplication in $\mathcal{R}_Q$ via fast NTT algorithms [31]. NTT maps a polynomial $\boldsymbol{c}(X) \in \mathcal{R}_Q$ from its coefficient vector $\boldsymbol{c}$ to an evaluation vector $\boldsymbol{a} = (a_0, \ldots, a_{N-1}) \in \mathbb{F}^N$, where $a_{\mathcal{Y}}$ evaluates $X = \omega^{2\mathcal{Y}+1}$ for $\mathcal{Y} \in [N]$. Since fast NTT produces $\boldsymbol{a}$ in bit-reversed order, we define $\mathsf{NTT}(\boldsymbol{c}) = \boldsymbol{a}^{\mathsf{R}}$ and its inverse as $\mathsf{INTT}(\boldsymbol{a}^{\mathsf{R}}) = \boldsymbol{c}$.

Extension Field. Let $F(X) \in \mathbb{F}[X]$ be an irreducible polynomial of degree D. By the Chinese Remainder Theorem, we define the extension field $\mathbb{F}[X]/(F)$, denoted by $\mathbb{EF}$, which has size Q^D. Each field element in $\mathbb{EF}$ is represented as a polynomial of degree at most $D-1$, and arithmetic within $\mathbb{EF}$ is performed using polynomial arithmetic modulo $F(X)$. Multilinear extensions of vectors and matrices can be similarly defined over $\mathbb{EF}$.

2.2. Fully Homomorphic Encryption

In this paper, we focus on the FHEW-like fully homomorphic encryption systems with fast bootstrapping procedures. An Learning with Errors (LWE) [34] ciphertext is of the form $(\boldsymbol{a}, b) = (\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + mq/t + e \mod q)$ where $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$, $s \in \{0,1\}^n$ is the secret key, $e \leftarrow \chi_\sigma$ is chosen from some discrete gaussian distribution with small norm, and $m \in \mathbb{Z}_t$ is the message. To define the relation among FHE various ciphertexts, we use $\mathbb{LWE}$ to denote the vector space $\mathbb{Z}_q^{n+1}$, allowing us to represent a LWE ciphertext as a vector $\mathsf{ct} = (\boldsymbol{a}, b) \in \mathbb{LWE}$.

A Ring Learning with Errors (RLWE) [27] ciphertext is of the form $(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}(X), \boldsymbol{a}(X) \cdot \boldsymbol{z}(X) + \boldsymbol{m}(x) + \boldsymbol{e}(X))$ where $\boldsymbol{a}(X) \leftarrow \mathcal{R}_Q$, $\boldsymbol{z}(X)$ is the secret polynomial, $\boldsymbol{e}(X) \leftarrow \chi_\sigma^N$ is sampled from discrete gaussian distribution, and $\boldsymbol{m}(X)$ is the message polynomial. Based on the form of RLWE ciphertexts, we use $\mathbb{RLWE}$ to denote the space $\mathbb{F}^N \times \mathbb{F}^N$, so that an RLWE ciphertext is represented by $\mathsf{c} \in \mathbb{RLWE}$, containting two vectors $\mathsf{c}.\boldsymbol{a} \in \mathbb{F}^N$ and $\mathsf{c}.\boldsymbol{b} \in \mathbb{F}^N$.

Let B is the basis for decomposition and $\ell = \lceil \log_B Q \rceil$. We define a RLWE$'$ ciphertext contains ℓ RLWE ciphertexts that encrypt the messages $\boldsymbol{m}, B\boldsymbol{m}, ..., B^{\ell-1}\boldsymbol{m}$ under the same secret key. Define a RGSW ciphertext contains the two RLWE$'$ ciphertexts that encrypt $-\boldsymbol{z}\boldsymbol{m}$ and $\boldsymbol{m}$, where $\boldsymbol{z}$ is the underlying secret polynomial. We use $\mathbb{RGSW}$ to denote the space $\mathbb{RLWE}^\ell \times \mathbb{RLWE}^\ell$, so we can represent an RGSW ciphertext $\mathsf{Z} \in \mathbb{RGSW}$ containting 4ℓ vectors $\mathsf{Z}.\boldsymbol{a}_0, \ldots, \mathsf{Z}.\boldsymbol{a}_{2\ell-1}$ and $\mathsf{Z}.\boldsymbol{b}_0, \ldots, \mathsf{Z}.\boldsymbol{b}_{2\ell-1}$. Let $\mathsf{Z}.\vec{\boldsymbol{a}} = (\mathsf{Z}.\boldsymbol{a}_0, \ldots, \mathsf{Z}.\boldsymbol{a}_{2\ell-1})$ and $\mathsf{Z}.\vec{\boldsymbol{b}} = (\mathsf{Z}.\boldsymbol{b}_0, \ldots, \mathsf{Z}.\boldsymbol{b}_{2\ell-1})$[1].

Next, we define operation $\mathbb{F}^N \otimes \mathbb{RLWE} \to \mathbb{RLWE}$: for a vector $\boldsymbol{d} \in \mathbb{F}^N$ and $\mathsf{c} \in \mathbb{RLWE}$, $\boldsymbol{d} \otimes \mathbb{RLWE} = (\mathsf{INTT}(\boldsymbol{d} \circ \mathsf{c}.\boldsymbol{a}), \mathsf{INTT}(\boldsymbol{d} \circ \mathsf{c}.\boldsymbol{b})) \in \mathbb{RLWE}$. We define operation $\mathbb{RLWE} \diamond \mathbb{RGSW} \to \mathbb{RLWE}$ between $\mathsf{c}' \in \mathbb{RLWE}$ and $\mathsf{Z} \in \mathbb{RGSW}$ as follows: we first decompose $\mathsf{c}'.\boldsymbol{a}$ and $\mathsf{c}'.\boldsymbol{b}$ into 2ℓ vectors, denoted by $\mathsf{bits}[0..2\ell] = (\mathsf{c}'.\boldsymbol{a}_0, \ldots, \mathsf{c}'.\boldsymbol{a}_{\ell-1}, \mathsf{c}'.\boldsymbol{b}_0, \ldots, \mathsf{c}'.\boldsymbol{b}_{\ell-1})$, and then we perform NTT on the small polynomials to obtain $\mathsf{Nbits}[0..2\ell]$, where each $\mathsf{Nbits}[i] = \mathsf{NTT}(\mathsf{bits}[i])$, and finally we compute

$$\left(\sum_{i=0}^{2\ell-1} \mathsf{Nbits}[i] \circ \mathsf{Z}.\boldsymbol{a}_i, \sum_{i=0}^{2\ell-1} \mathsf{Nbits}[i] \circ \mathsf{Z}.\boldsymbol{b}_i \right) \in \mathbb{RLWE}$$

This operation is often called as external product.

The core operation that we prove in this article is the heaviest bootstrapping procedure. We provide a brief overview of the entire process and refer the reader to [35] for detailed explanations.

Given two LWE ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$ encrypt two bits with binary secret key $\boldsymbol{s} = (s_0, ..., s_{n-1})$. We first add these two ciphertexts into a new ciphertext $\mathsf{ct} = (\boldsymbol{a}, b) \in \mathbb{Z}_q^{n+1}$ by adding the components of each LWE ciphertext in $\mathbb{Z}_q$.

In the next step, the components of ct are exponentiated by the polynomial $Y = X^{2N/q}$. Specifically, we compute $Y^{-a_0}, \ldots, Y^{-a_{n-1}}, Y^b$. We assume that $2N$ is divisible by q, which is a standard parameter choice in FHE schemes. This process is referred to as the lift procedure.

Let $\{\mathsf{Z}_i\}_{0 \le i \le n-1}$ be the bootstrapping key, which are RGSW ciphertexts encrypting s_i. Let $\boldsymbol{v}$ be a constant polynomial used to facilitate the extraction of an LWE ciphertext; its specific form is omitted here as it is not crucial to our design. The accumulator, initialized as $\mathsf{acc}[0] = \boldsymbol{v} \cdot Y^b$ is iteratively updated as $\mathsf{acc}[i+1] = \mathsf{acc}[i] + Y^{-a_i} \otimes \mathsf{acc}[i] \diamond \mathsf{Z}_i$ for $0 \le i \le n-1$. At the end of this process, the result is an RLWE ciphertext encrypting $\boldsymbol{v} \cdot Y^{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle}$.

Let $(\boldsymbol{a}', \boldsymbol{b}')$ be the resulting RLWE ciphertext. The coefficient vector of $\boldsymbol{a}'$ and the constant term of $\boldsymbol{b}'$ are extracted as an LWE ciphertext defined over $\mathbb{Z}_Q$. To convert this into a ciphertext defined over $\mathbb{Z}_q$, a modulus switching procedure is applied to each component. The modulus switching function is defined as: $x \in \mathbb{Z}_Q \to \lfloor \frac{x \cdot q}{Q} \rceil \mod q$.

We note a minor modification from the standard bootstrapping procedure. In our design, we set $n = N$ and define the secret key $\boldsymbol{z}$ in the bootstrapping key as a reordered version of the vector $\boldsymbol{s}$. This avoids the need for a key switching step, simplifying the overall process. It is straightforward to adapt this design to the standard bootstrapping procedure.

2.3. Useful PIOP

Sumcheck Protocol. We describe a seminal interactive proof used in our work, known as the sumcheck protocol,

[1]When we say $\mathsf{Z} \in \mathbb{RGSW}$, we intend to represent an RGSW ciphertext in its NTT form rather than in its coefficient form.

proposed by Lund et al.[36]. Suppose there is an μ-variate low-degree polynomial $\mathcal{G} : \mathbb{F}^\mu \to \mathbb{F}$ where the degree of each variable in $\mathcal{G}$ is at most ℓ. A verifier $\mathcal{V}$, wishes to verify the following claim by an untrusted prover $\mathcal{P}$:

$$T = \sum_{x \in \{0,1\}^\mu} \mathcal{G}(x_1, x_2, \ldots, x_\mu)$$

In the sum-check protocol, $\mathcal{V}$ interacts with $\mathcal{P}$ over μ rounds. At the end of this interaction, $\mathcal{V}$ outputs $b \in \{0,1\}$. The principal cost to $\mathcal{V}$ is the evaluation of $\mathcal{G}$ at a random point in its domain $r \in \mathbb{F}^\mu$. Throughout this paper, we consider the protocol run over the extension field $\mathbb{EF}$, where verifier $\mathcal{V}$ evaluates $\mathcal{G}$ at a random point $r \in \mathbb{EF}^\mu$. We treat the sumcheck protocol as a mechanism to reduce the claim $\sum_{x \in \{0,1\}^\mu} \mathcal{G} \stackrel{?}{=} T$ to the claim $\mathcal{G}(r) \stackrel{?}{=} e$ and denote this reduction protocol as $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T)$, with the following properties:

- **Completeness.** If $T = \sum_{x \in \{0,1\}^\mu} \mathcal{G}(x)$, then for a correct $\mathcal{P}$ and for all $r \in \{0,1\}^*$, $\Pr[\langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T) = 1] = 1$.
- **Soundness.** If $T \neq \sum_{x \in \{0,1\}^\mu} \mathcal{G}(x)$, then for any $\mathcal{P}^*$ and for all $r \in \{0,1\}^*$, $\Pr[\langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T) = 1] \leq \ell \cdot \mu / |\mathbb{EF}|$.
- **Succinctness.** The proof size is $O(\mu \cdot \ell)$ elements of $\mathbb{EF}$.

The prover time is linear to the time required to evaluate the sum, while the verifier time is $O(\mu)$, plus the cost of evaluating $\mathcal{G}$ at a random point, which corresponds to the query complexity in the polynomial IOP (PIOP).

Hadamard Protocol. We consider a Hadamard protocol as a PIOP for the index relation $\mathcal{R}_\circ$, which proves a sum of the Hadamard products. The detailed construction can be easily achieved using the sumcheck protocol, shown in Construction 1,ensuring perfect completeness and a soundness error of $\frac{O(\log N)}{|\mathbb{EF}|}$ with $O(\log N)$ round complexity. The prover can be implemented in $O(MN)$ field operations, while the verifier can be implemented in $O(\log N + M)$.

Definition 1. *The indexed Hadamard relation $\mathcal{R}_\circ$ is the set of tuples* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) =$

$$(\perp, (\mathbb{F}, \mathbb{EF}, N, M), (\{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{M-1}\}, \{\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{M-1}\}, \boldsymbol{c}))$$

where $\forall i, \boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c} \in \mathbb{F}^N$ *and* $\sum_{i=0}^{M-1} \boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}$.

Remark 1. *The witness* $\boldsymbol{c}$ *can be* $\mathbf{0}$.

Construction 1. *We construct a PIOP for the indexed relation $\mathcal{R}_\circ$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, M)$, and witness $\mathbb{w} = (\{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{M-1}\}, \{\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{M-1}\}, \boldsymbol{c})$.*

- *The verifier $\mathcal{V}$ samples a challenge $u \xleftarrow{\$} \mathbb{EF}^{\log N}$ and sends it to $\mathcal{P}$.*
- *Let $\mathcal{G}(x) = \tilde{eq}(x, u) \cdot (\sum_{i=0}^{M-1} \boldsymbol{a}_i(x) \cdot \boldsymbol{b}_i(x) - \boldsymbol{c}(x)), T = 0, \mu = \log N, \ell = 3$.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the sumcheck protocol $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T)$*
- *The verifier $\mathcal{V}$ makes the evaluation queries for $i \in [M]$:*
$$e^{\boldsymbol{a}_i} := \boldsymbol{a}_i(r), \quad e^{\boldsymbol{b}_i} := \boldsymbol{b}_i(r)$$
and $e^{\boldsymbol{c}} := \boldsymbol{c}(r)$.
check that $\tilde{eq}(x, r_i) \cdot (\sum_{i=0}^{M-1} e^{\boldsymbol{a}_i} \cdot e^{\boldsymbol{b}_i} - e^{\boldsymbol{c}}) = e$.

Lookups Based on the Logarithmic Derivative. Lookup arguments are used to prove that a sequence of values belongs to a table, which is often prescribed. Haböck proposed a lookup argument based on the logarithmic derivative in [37]. Let N be a power-of-two less than the characteristic of field $\mathbb{F}$, i.e. $N < Q$. Given two vectors of field elements $\boldsymbol{a} \in \mathbb{F}^N$ and $\boldsymbol{t} \in \mathbb{F}^M$, lookup problem can be written as $\{a_i : i = 1, \ldots, N\} \subseteq \{t_j : j = 1, \ldots, M\}$ as sets, with multiples of values removed. For simplicity, we assume $M = N$. According to the Lemma 5 of [37], we have $\{a_i\} \subseteq \{t_i\}$ as sets if and only if there exists a vector $\boldsymbol{m} \in \mathbb{F}^N$ such that

$$\sum_{i=1}^{N} \frac{1}{X + a_i} = \sum_{i=1}^{N} \frac{m_i}{X + t_i}$$

in the function field $\mathbb{F}[X]$. Let $\mathsf{mp}(\boldsymbol{a}, z)$ denote the multiplicity of an element z in the vector $\boldsymbol{a}$, with a similar notation for $\boldsymbol{t}$. The vector $\boldsymbol{m}$ can be computed as $\boldsymbol{m}_i = \frac{\mathsf{mp}(\boldsymbol{a}, t_i)}{\mathsf{mp}(\boldsymbol{t}, t_i)}$ in probablisitic polynomial time.

We consider the batch lookup arguments proposed in [37]. Let $\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{M-1}$ and $\boldsymbol{t}$ be multivariate functions defined over the Boolean hypercube, i.e., $\{0,1\}^{\log N} \to \mathbb{F}$. By Lemma 5 in [37], we have $\bigcup_{i=0}^{L-1} \{\boldsymbol{f}_i(x)\} \subseteq \{\boldsymbol{t}(x)\}$ as sets if and only if there exists a function $\boldsymbol{m} : \{0,1\}^{\log N} \to \mathbb{F}$ such that

$$\sum_{x \in \{0,1\}^{\log N}} \sum_{i=0}^{M-1} \frac{1}{X + \boldsymbol{f}_i(x)} = \sum_{x \in \{0,1\}^{\log N}} \frac{\boldsymbol{m}(x)}{X + \boldsymbol{t}(x)}$$

in the function field $\mathbb{F}[X]$. For each $x \in \{0,1\}^{\log N}$, the function $\boldsymbol{m}(x)$ can be computed as $\boldsymbol{m}(x) = \frac{\sum_{i=0}^{M-1} \mathsf{mp}(\boldsymbol{f}_i, \boldsymbol{t}(x))}{\mathsf{mp}(\boldsymbol{t}, \boldsymbol{t}(x))}$.

Hence, we have a PIOP described in Construction 2 for the following relation $\mathcal{R}_{\mathsf{lookup}}$ that has the soundness error $O\left(\frac{MN}{|\mathbb{EF}|}\right)$. The prover can be implemented in $O(MN)$ and the verifier time is $O(\log N + M)$.

Definition 2. *The batch lookups relation $\mathcal{R}_{\mathsf{lookup}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}, M), (\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{M-1}))$$

where $\boldsymbol{t}, \boldsymbol{f} : \{0,1\}^{\log N} \to \mathbb{F}$ and $\bigcup_{i=0}^{M-1} \{\boldsymbol{f}_i(x)\} \subseteq \{\boldsymbol{t}(x)\}$ as sets.

Construction 2. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{lookup}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}, M)$, and witness $\mathbb{w} = (\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{M-1})$.*

- *$\mathcal{P}$ computes the multiplicity polynomial $\boldsymbol{m}(x) = \frac{\sum_{i=0}^{M-1} \mathsf{mp}(\boldsymbol{f}_i,)}{\mathsf{mp}(\boldsymbol{t}, \boldsymbol{t}(x))}$ and sends the oracle message to the verifier.*
- *The verifier $\mathcal{V}$ samples a challenge $\rho \xleftarrow{\$} \mathbb{EF}$ uniformly at random and sends it to the prover $\mathcal{P}$.*
- *The prover $\mathcal{P}$ compute $\boldsymbol{h}_i(x) = \frac{\boldsymbol{m}_i(x)}{\boldsymbol{\varphi}_i(x)}$ where $\boldsymbol{m}_i(x) = -1$ and $\boldsymbol{\varphi}_i(x) = \rho + \boldsymbol{f}_i(x)$ for $i \in [M]$ and $\boldsymbol{m}_M(x) =$*

$\boldsymbol{m}(x)$ and $\boldsymbol{\varphi}_L(x) = \rho + \boldsymbol{t}(x)$. The prover sends the oracle messages $\boldsymbol{h}_0, \ldots, \boldsymbol{h}_M$ to the verifier.
- *Let $\mathcal{G} = \sum_{i=0}^{M} \boldsymbol{h}_i(x), T = 0, \mu = \log N, \ell = 1$.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the sumcheck protocol $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T)$ where $r \in \mathbb{EF}^{\mu}$.*
- *The verifier $\mathcal{V}$ makes the evaluation query for $i \in [L+1]$:*
$$e^{\boldsymbol{h}_i} = \boldsymbol{h}_i(r)$$
check that $\sum_{i=0}^{M} e^h i = e$.
- *For $i \in [M+1]$, the prover and the verifier run the Hadamard protocol in parallel with*
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{h}_i, \boldsymbol{\varphi}_i, \boldsymbol{m}_i))$

For convenience, we also extend it to a PIOP for the indexed lookup relation $\mathcal{R}_{\mathsf{idx-lookup}}$, which adds a column specifying the table index.

Definition 3. *The indexed lookups relation $\mathcal{R}_{\mathsf{idx-lookup}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}), (\mathsf{idx}, \boldsymbol{f}))$$

where $\boldsymbol{t}, \boldsymbol{f} \in \mathbb{F}^N$, $\mathsf{idx} \in \mathbb{Z}_N^N$ and $\{(\mathsf{idx}(x), \boldsymbol{f}(x))\} \subseteq \{(\mathsf{to\text{-}field}(x), \boldsymbol{t}(x))\}$ as sets.

3. PIOP for Building Blocks

Before introducing the PIOP for FHE operations, we first propose our PIOP for key atomic operations.

3.1. Sparse Matrix Evaluation

[37][28] proposed a standard commitment scheme for sparse multilinear polynomials with optimal prover costs, where the sparse matrices are represented by dense polynomials. The evaluation proof in this case is essentially a specialized PIOP with dense polynomials committed. In this section, we consider a specialized sparse $N \times M$ matrix C, where each column contains exactly one non-zero entry. Building on Claim 1 in [37] for the evaluation of general sparse matrices, we define the evaluation for this specialized sparse matrix in Lemma 1, and propose a PIOP for the relation $\mathcal{R}_{\mathsf{sparse}}$ in Definition 4, outlined in Construction 3.

Lemma 1. *Given a $\log N + \log M$-variate multilinear polynomial $\tilde{C}$ defined over a sparse $N \times M$ matrix C, where each column has only one non-zero value, there exist two $(\log M)$-variate multilinear polynomials $\mathsf{row}, \mathsf{val}$ such that the following holds for all $r_x \in \mathbb{F}^{\log N}, r_y \in \mathbb{F}^{\log M}$.*

$$\tilde{C}(r_x, r_y) = \sum_{k \in \{0,1\}^{\log M}} \mathsf{val}(k) \cdot \tilde{eq}\left(\mathsf{to\text{-}bits}(\mathsf{row}(k)), r_x\right) \cdot \tilde{eq}\left(k, r_y\right)$$

Proof. The proof follows the Claim 1 in [37] with $\mathsf{col}(k) = \mathsf{to\text{-}field}(k)$ for all $k \in \{0,1\}^{\log M}$. □

Definition 4. *The evaluation relation $\mathcal{R}_{\mathsf{sparse}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, M, e, r_x, r_y), (\mathsf{val}, \mathsf{row}))$$

where $r_x \in \mathbb{EF}^{\log N}$, $r_y \in \mathbb{EF}^{\log M}$, $e \in \mathbb{EF}$, and $\mathsf{val} \in \mathbb{F}^M$, $\mathsf{row} \in \mathbb{Z}_M^M$, and $\sum_{k \in \{0,1\}^{\log M}} \mathsf{val}(k) \cdot \tilde{eq}\left(\mathsf{to\text{-}bits}(\mathsf{row}(k)), r_x\right) \cdot \tilde{eq}\left(k, r_y\right) = e$.

Theorem 1. *For every finite field $\mathbb{F}$ and extension field $\mathbb{EF}$, and positive integers N, M, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{sparse}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $\frac{O(M)}{|\mathbb{EF}|}$ with $O(\log M)$ round complexity. The prover can be implemented in $O(M)$ extension field operations, and the verifier can be implemented in $O(\log M)$.*

We prove it with the following Construction 3.

Construction 3. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{sparse}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, M, e, r_x, r_y)$, and witness $\mathbb{w} = (\mathsf{val}, \mathsf{col})$; the verifier $\mathcal{V}$ takes as input the index $\mathbb{i}$and the instance $\mathbb{x}$.*
- *For $k \in \{0,1\}^{\log M}$: the prover $\mathcal{P}$ computes $E_{r_x}(k) = \tilde{eq}\left(\mathsf{to\text{-}bits}(\mathsf{row}(k)), r_x\right)$.*
- *Let $\mathcal{G}(k) = \mathsf{val}(k) \cdot E_{r_x}(k) \cdot \tilde{eq}(k, r_y), T = e, \mu = \log M, \ell = 3$.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the sumcheck protocol $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T)$ where $r \in \mathbb{EF}^{\log M}$.*
- *The verifier $\mathcal{V}$ makes the evaluation query for val, E_{r_x}:*
$$e^{\mathsf{val}} := \mathsf{val}(r), \quad , e^{E_{r_x}} := E_{r_x}(r)$$
checks that $e^{\mathsf{val}} \cdot e^{E_{r_x}} \cdot \tilde{eq}(r, r_y) = e$.
- *Let $\boldsymbol{t}(x) = \tilde{eq}(x, r_x)$ for $x \in \{0,1\}^{\log N}$ and $\mathsf{idx}(k) = \mathsf{row}(k)$ for $k \in \{0,1\}^{\log M}$.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ check that E_{r_x} is well-formed using indexed lookup with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}), (\mathsf{idx}, E_{r_x}))$*

Remark 2. *Instead of using a standard memory-checking technique [38] to prove E_{r_x} is well-formed as described in [28][29], we use an indexed lookup protocol to prove the values E_{r_x}, along with the index row, are prescribed in the table $\boldsymbol{t} = \tilde{eq}(\cdot, r_x)$. This simplification is feasible because the table size N is a small parameter in FHE.*

3.2. NTT/INTT

In this section, we propose a specialized PIOP for the $\mathsf{NTT}/\mathsf{INTT}$ operation, where the evaluation vector is arranged in a bit-reversed order. Our protocol is inspired by the approach in [30] to prove FFT. Additionally, we propose an optimization for proving batched NTT operations, leveraging the linearity of the NTT operation. Specifically, this optimization provides asymptotical improvements when the coefficient vectors are sparse; it also concretely saves the computational costs when the proof system operates in the extension field $\mathbb{EF}$, while the coefficient vectors are defined over the base field $\mathbb{F}$.

In our work, the NTT operation transforms the polynomial's coefficient vector $\boldsymbol{c} \in \mathbb{F}^N$ into an evaluation vector $\boldsymbol{a} \in \mathbb{F}^N$ at $X = \omega^{2\mathcal{Y}+1}$ for $\mathcal{Y} \in [N]$ while the output vector is arranged in a bit-reversed order, i.e. $\boldsymbol{a}^{\mathsf{R}}$. Specifically, the $\mathcal{Y}$-th entry of $\boldsymbol{a}^{\mathsf{R}}$ represents the evaluation of $X = \omega^{2\mathcal{Y}^{\mathsf{R}}+1}$.

Hence, we can express the NTT operation as a matrix-vector multiplication $\boldsymbol{a}^{\mathsf{R}} = F^{\mathsf{R}} \cdot \boldsymbol{c}$, where F^{R} is a Fourier matrix defined by $F^{\mathsf{R}}(\mathcal{Y}, \mathcal{X}) = \omega^{(2\mathcal{Y}^{\mathsf{R}}+1)\cdot\mathcal{X}}$. To prove this matrix-vector multiplication, we first turn the equation of polynomial evaluation to the form of multivariate polynomials:

$$\tilde{\boldsymbol{a}}^{\mathsf{R}}(y) = \sum_{x\in\{0,1\}^{\log N}} \tilde{\boldsymbol{c}}(x)\tilde{F}^{\mathsf{R}}(y,x) \quad (1)$$

for $y \in \{0,1\}^{\log N}$. To run the sumcheck protocol on equation 1, we use the algorithm in [33] and [12]. Given the evaluation $\tilde{\boldsymbol{a}}^{\mathsf{R}}(u)$ at a random point $u \in \mathbb{EF}^{\log N}$, [33] provides a dynamic programming algorithm for the prover to initialize the values of $\tilde{\boldsymbol{c}}$(x) on all $x \in \{0,1\}^{\log N}$ in linear time, where the initialization is referred as the bookkeeping table in [12]. Prior work [30] presents a linear-time algorithm to compute $\tilde{F}(u,x)$ for proving FFT, where the Fourier matrix is defined by $F(\mathcal{Y},\mathcal{X}) = \omega^{\mathcal{Y}\mathcal{X}}$ for $\mathcal{Y} \in [M], \mathcal{X} \in [N]$, so this approach cannot be directly applied to compute $\tilde{F}^{\mathsf{R}}(u,x)$ when proving fast NTT.

Instead, we adopt a similar approach by decomposing the exponents of the roots of unity ω. The term $\omega^{2^{\log N-i}} = \omega^{\frac{2N}{2^{i+1}}}$ corresponds to the 2^{i+1}-th roots of unity, denoted as $\omega_{2^{i+1}}$, where $\omega_{i+1}^{\mathcal{X}}$ takes on 2^{i+1} distinct values for all $\mathcal{X} \in [N]$. We write $\tilde{F}^{\mathsf{R}}(u,x)$ as follows:

$$\begin{aligned}
\tilde{F}^{\mathsf{R}}(u,x) &= \sum_{y\in\{0,1\}^{\log N}} \tilde{eq}\,(u,y)\,\tilde{F}^{\mathsf{R}}(y,x)\\
&= \sum_{y\in\{0,1\}^{\log N}} \tilde{eq}\,(u,y)\,\omega^{(2\mathcal{Y}^{\mathsf{R}}+1)\cdot\mathcal{X}}\\
&= \omega^{\mathcal{X}} \cdot \sum_{y\in\{0,1\}^{\log N}} \tilde{eq}\,(u,y)\,\omega^{\mathcal{X}\cdot 2\mathcal{Y}^{\mathsf{R}}} \quad (2)\\
&= \omega^{\mathcal{X}} \cdot \sum_{y\in\{0,1\}^{\log N}} \tilde{eq}\,(u,y)\,\omega^{\mathcal{X}\cdot(\sum_{i=0}^{\log N-1} 2^{\log N-i}\cdot y_i)}\\
&= \omega^{\mathcal{X}} \cdot \sum_{y\in\{0,1\}^{\log N}} \prod_{j=0}^{\log N-1} ((1-u_j)(1-y_j)+u_jy_j) \cdot \prod_{i=0}^{\log N-1} \left(\omega^{2^{\log N-i}}\right)^{\mathcal{X}\cdot y_i}\\
&= \omega^{\mathcal{X}} \cdot \sum_{y\in\{0,1\}^{\log N}} \prod_{i=0}^{\log N-1} ((1-u_i)(1-y_i)+u_iy_i)\cdot\omega_{2^{i+1}}^{\mathcal{X}\cdot y_i}\\
&= \omega^{\mathcal{X}} \cdot \prod_{i=0}^{\log N-1} \sum_{y_i\in\{0,1\}} ((1-u_i)(1-y_i)+u_iy_i)\cdot\omega_{2^{i+1}}^{\mathcal{X}\cdot y_i}\\
&= \omega^{\mathcal{X}} \cdot \prod_{i=0}^{\log N-1} (1-u_i+u_i\cdot\omega_{2^{i+1}}^{\mathcal{X}}) \quad (3)\\
&= \omega^{\sum_{j=0}^{\log N-1} 2^j\cdot x_j} \cdot \prod_{i=0}^{\log N-1} (1-u_i+u_i\cdot\omega_{2^{i+1}}^{\mathcal{X}})\\
&= \prod_{i=0}^{\log N-1} (1-u_i+u_i\cdot\omega_{2^{i+1}}^{\mathcal{X}})\cdot\omega^{2^i\cdot x_i} \quad (4)
\end{aligned}$$

Here, we decompose the bits of $\mathcal{Y}^{\mathsf{R}}$ in Equation 2 and decompose the bits of $\mathcal{X}$ in Equation 3. Note that Equation 4 includes an extra term, $\omega^{2^i\cdot x_i}$, in the grand product compared to Equation (8) in [30], due to the different structure of the Fourier matrix used for proving NTT.

Using the property that $\omega_{2^{i+1}}$ has 2^i+1 distinct values, we can divide the computation into $\log N$ rounds via dynamic programming, as described in the Algorithm 1. In particular, Step 4 of Algorithm 1 computes the most significant bit (MSB) of j, which indicates x_i for i-th round, with i updating from 0 to $\log N - 1$.

Remark 3. *An alternative method for the prover to compute* $\tilde{F}^{\mathsf{R}}(u,x) = \omega^{\mathcal{X}} \cdot \prod_{i=0}^{\log N-1}(1-u_i+u_i\cdot\omega_{2^{i+1}}^{\mathcal{X}})$ *is to multiply by* $\omega^{\mathcal{X}}$ *after performing the dynamic algorithm. However, to reduce the verifier time through delegating the computation of* $\tilde{F}(u,v)$*, the delegation protocol described in the following paragraph has to follow Algorithm 1.*

Algorithm 1 $A_{F^{\mathsf{R}}} \leftarrow \mathsf{Initialize}(\omega,\mu,N)$

```
Input: The degree $N$ and the $2N$-th root of unity $\omega$.
Input: The random point $u \in \mathbb{EF}^{\log N}$.
Output: $A_{F^{\mathsf{R}}}$ storing $\tilde{F}^{\mathsf{R}}(u,x)$ for all $x \in \{0,1\}^{\log N}$
1: $A_{F^{\mathsf{R}}}[0] = 1$
2: for $i = 0,\ldots,\log N-1$ do
3:     for $j = 2^{i+1}-1,\ldots,0$ do
4:         $b = (j >> i)\&1$    // the MSB of $j$
5:         $A_{F^{\mathsf{R}}}[j] = A_{F^{\mathsf{R}}}[j\%2^i]\cdot((1-u_i)+u_i\cdot\omega_{2^{i+1}}^{j})\cdot\omega^{2^i\cdot b}$
6: return $A_{F^{\mathsf{R}}}$
```

Reducing the verifier time. At the last round of the sumcheck protocol, the verifier needs to evaluate $\tilde{F}^{\mathsf{R}}(u,\cdot)$ at a random point $v \xleftarrow{\$} \mathbb{EF}^{\log N}$. While the algorithm for computing $\tilde{F}^{\mathsf{R}}(u,v)$ differs from [30], we adopt a similar delegation protocol to reduce the verifier time to $O(\log^2 N)$.

The evaluation for $\tilde{F}^{\mathsf{R}}(u,v)$ can be delegated to the prover through a sequence of sumcheck protocol, following the same Algorithm 1 to compute $A_{F^{\mathsf{R}}}$. We abuse the notation and use $A_{F^{\mathsf{R}}}^{(i)} : \{0,1\}^{i+1} \to \mathbb{EF}$ as in [30] to denote the array $A_{F^{\mathsf{R}}}$ in the i-th round for $i = 0,\ldots,\log N-1$. Then $\tilde{F}^{\mathsf{R}}(u,v) = \tilde{A}_{F^{\mathsf{R}}}^{(\log N-1)}(v)$, and we can write $A_{F^{\mathsf{R}}}^{(i)}(\cdot)$ as an equation of $A_{F^{\mathsf{R}}}^{(i-1)}(\cdot)$:

$$A_{F^{\mathsf{R}}}^{(i)}(x,b) = A_{F^{\mathsf{R}}}^{(i-1)}(x)\,((1-u_i)+u_i\cdot\omega_{i+1}(x,b))\cdot\omega^{2^i\cdot b}$$

for all $x \in \{0,1\}^i, b \in \{0,1\}$, where $\omega_{i+1}(x,b) = \omega_{2^{i+1}}^{j}$ with $j = \mathcal{X} + 2^i \cdot b$. Note that the right-hand side of the above equation is not multilinear in either x or b, so we introduce a $(i+1)$-variate identity function to obtain the following multilinear function

$$\begin{aligned}
&\tilde{A}_{F^{\mathsf{R}}}^{(i)}(x,b)\\
&= \sum_{z\in\{0,1\}^i}\sum_{s\in\{0,1\}} \tilde{eq}\,((x,b),(z,s))\cdot\tilde{A}_{F^{\mathsf{R}}}^{(i-1)}(z)\\
&\qquad\cdot((1-u_i)+u_i\cdot\tilde{\omega}_{i+1}(z,s))\cdot\omega^{2^i\cdot s}
\end{aligned}$$

that holds for all $x \in \mathbb{EF}^i, b \in \mathbb{EF}$.

Starting from $\tilde{F}^{\mathsf{R}}(u,v) = \tilde{A}_{F^{\mathsf{R}}}^{(\log N-1)}(v)$, the verifier and the prover can reduce its correctness to the evaluation of $\tilde{A}_{F^{\mathsf{R}}}^{(i)}(\cdot)$ at a random point through a sumcheck protocol for $i = \log N - 1, \ldots, 0$. In the last round as defined in the Step 1 of Algorithm 1, $\tilde{A}_{F^{\mathsf{R}}}^{(0)}(\cdot)$ is simply the constant 1. At the end of each sumcheck protocol, the verifier has to evaluate $\tilde{eq}(\cdot)$ and $\tilde{\omega}_{i+1}$ at a random point to obtain $\tilde{A}_{F^{\mathsf{R}}}^{(i-1)}$ at the random point for the next sumcheck protocol. By the closed-form definition of multilinear extension, we have $\tilde{\omega}_{i+1}(r) = \sum_{x \in \{0,1\}^{i+1}} \tilde{eq}(r,x)\, \omega_{2^{i+1}}^{j}$ for $j = \sum_{k=0}^{i+1} 2^k \cdot x_k$, which equals to $\prod_{k=0}^{i+1}(1 - r_k + r_k \cdot \omega_{2^{i+1}}^{2^k})$.

By delegating the computation of $\tilde{F}^{\mathsf{R}}(u,v)$, the verifier only needs to evaluate $\tilde{\boldsymbol{c}}(\cdot)$ at the random point v. Consequently, we view the entire PIOP (including the delegation protocol) for proving NTT relation $\mathcal{R}_{\mathsf{NTT}}$ as a mechanism to reduce a claim of the form $\sum_{x \in \{0,1\}^{\log N}} \tilde{\boldsymbol{c}}(x) \cdot \tilde{F}^{\mathsf{R}}(u,x) \overset{?}{=} \tilde{\boldsymbol{a}}^{\mathsf{R}}(u)$ to a subclaim $d \overset{?}{=} \tilde{\boldsymbol{c}}(v)$, where $v \xleftarrow{\$} \mathbb{EF}^{\log N}$.

Definition 5. *The NTT relation $\mathcal{R}_{\mathsf{NTT}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, u, e), (\boldsymbol{c}))$$

where $u \in \mathbb{EF}^N, e \in \mathbb{EF}, \boldsymbol{c} \in \mathbb{EF}^N$, and $e = \sum_{x \in \{0,1\}^{\log N}} \boldsymbol{c}(x) \cdot \tilde{F}^{\mathsf{R}}(u,x)$.

Theorem 2. *For every finite field $\mathbb{F}$ and extension field $\mathbb{EF}$, and positive integers N, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{NTT}}$ that supports instances over $\mathbb{EF}$, with perfect completeness and soundness error $\frac{O(\log N)}{|\mathbb{EF}|}$ with $O(\log^2 N)$ round complexity. The prover can be implemented in $O(N)$ field operations, and the verifier can be implemented in $O(\log^2 N)$.*

Proof. Theorem 2 can be proven with construction that consists of a sumcheck protocol (with the bookkeeping table for $\tilde{F}^{\mathsf{R}}(u,x)$ provided using Algorithm 1) and a delegation protocol. □

Batched NTT Operation. Consider batches of NTT instances with coefficient vectors $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1} \in \mathbb{F}^N$ and their evaluations $\tilde{\boldsymbol{a}}_0^{\mathsf{R}}(u), \ldots, \tilde{\boldsymbol{a}}_{M-1}^{\mathsf{R}}(u) \in \mathbb{EF}$ at the same random point $u \xleftarrow{\$} \mathbb{EF}^{\log N}$. By the linearity of the NTT operation, we can compute a randomized NTT instance with a randomized coefficient vector $\boldsymbol{c}' = \sum_{i=0}^{M-1} \rho_i \cdot \boldsymbol{c}_i \in \mathbb{EF}$ and its evaluation $e = \sum_{i=0}^{M-1} \rho_i \cdot \tilde{\boldsymbol{a}}_i^{\mathsf{R}}(u) \in \mathbb{EF}$ at the point u, where $\rho_0, \ldots, \rho_{M-1} \xleftarrow{\$} \mathbb{EF}$. The correctness follows from the linearity of NTT operation that $\sum_{i=0}^{M-1} \rho_i \cdot {\boldsymbol{a}_i}^{\mathsf{R}} = \sum_{i=0}^{M-1} \rho_i \cdot F^{\mathsf{R}} \boldsymbol{c} = F^{\mathsf{R}} \sum_{i=0}^{M-1} \rho_i \cdot \boldsymbol{c}_i = \mathsf{NTT}(\boldsymbol{c}')$. Hence, the prover time for proving batched NTT operations consists of the computation of the randomized coefficient vector and proving a single randomized NTT instance, where the former dominates. This process leads to a concrete computational optimization for the prover, reducing $O(MN)$ extension field operations to $O(MN)$ multiplications between the base field and the extension field.

Additionally, if the coefficient vectors are sparse with only $O(M)$ non-zero entries, the prover time can be reduced to $O(M+N)$. This optimization is used in our PIOP for proving batched lift, which is described in Section 4.2.

3.3. Gadget Decomposition

In this section, we introduce a PIOP for the operation $\mathsf{Dec}(\boldsymbol{a}; B, \ell)$ that decomposes a vector $\boldsymbol{a} \in \mathbb{F}^N$ into ℓ gadgets $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1} \in \mathbb{Z}_B^N$ such that $\boldsymbol{a} = \sum_{i=0}^{\ell-1} B^i \cdot \boldsymbol{a}_i$ where $\ell = \lceil \log_B Q \rceil$. We utilize the PIOP for proving the batched lookup relation $\mathcal{R}_{\mathsf{lookup}}$ to check that the range of decomposed gadgets is contained within the table $\boldsymbol{t} = (0, \ldots, B-1)$.

Definition 6. *The decomposition relation $\mathcal{R}_{\mathsf{dec}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell), (\boldsymbol{a}, (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1})))$$

where $\ell = \lceil \log_B Q \rceil$, $\boldsymbol{a} \in \mathbb{F}^N$, $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1} \in \mathbb{Z}_B^N$ and $\boldsymbol{a} = \sum_{i=0}^{\ell-1} B^i \cdot \boldsymbol{a}_i$.

Theorem 3. *For every finite field $\mathbb{F}$ and extension field $\mathbb{EF}$, and positive integers N, M, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{dec}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $\frac{O(\ell N)}{|\mathbb{EF}|}$ with $O(\log N)$ round complexity. The prover can be implemented in $O(\ell N)$ field operations, and the verifier can be implemented in $O(\log N + \ell)$.*

We prove it with the following Construction 5.

Construction 4. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{dec}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, B, \ell)$, and witness $\mathbb{w} = (\boldsymbol{a}, (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1}))$.*

- *The prover $\mathcal{P}$ and verifier $\mathcal{V}$ sets $\boldsymbol{t} = (0, \ldots, B-1, 0, \ldots, 0) \in \mathbb{F}^N$ and run the batched lookup protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, N, \boldsymbol{t}), (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1}))$.*
- *The verifier samples a random point on $r \xleftarrow{\$} \mathbb{EF}^{\log N}$ and makes the queries for $\boldsymbol{a}$, $\boldsymbol{a}_0$, …, $\boldsymbol{a}_{\ell-1}$.*
  $e^{\boldsymbol{a}} = \boldsymbol{a}(r), \quad e^{\boldsymbol{a}_0} = \boldsymbol{a}_0(r), \quad \ldots, \quad e^{\boldsymbol{a}_{\ell-1}} = \boldsymbol{a}_{\ell-1}(r)$
  check that $\sum_{i=0}^{\ell-1} B^i \cdot e^{\boldsymbol{a}_i} = e^{\boldsymbol{a}}$.

Extension to Range Check. In this paper, we generally use the lookup protocol to check $a \in [0, L)$ with a table $\boldsymbol{t} = (0, \ldots, L-1)$ when L is relatively small. However, when $L = O(Q)$ is large, approaching the size of $\mathbb{F}$, we instead employ the method of gadget decomposition to check the range relation $\mathcal{R}_{\mathsf{range}}$ in Definition 7, as the prover time of lookup arguments increases linearly with the table size. If the range size L is not an exact power of B, and the next power of B greater than L is B^ℓ, the prover can compute the decomposed gadgets of $\boldsymbol{a}$ and $\boldsymbol{a} + (B^\ell - L)$ with two invocations for proving $\mathcal{R}_{\mathsf{dec}}$, ensuring that the intersection of the two proved ranges is $[0, L)$. The PIOP construction for $\mathcal{R}_{\mathsf{range}}$ is omitted here.

Definition 7. *The range check relation $\mathcal{R}_{\mathsf{range}}$ is the set*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell, L), \boldsymbol{a})$$

where B^ℓ is the next exact power of B greater than or equal to L and $\boldsymbol{a} \in \mathbb{Z}_L^N$.

4. PIOP for FHE Operations

In this section, we introduce our PIOP for operations defined in the FHE NAND circuit as described in Fig. 1.

4.1. LWE Addition

The core of proving LWE addition is simulating a straightforward addition modulo q in our proof systems, which uses modulus Q. Given $a, b \in \mathbb{Z}_q$, the result $c = a+b \mod q$ holds if and only if there exists $k \in \{0,1\}$ such that $c \in \mathbb{Z}_q$ and $a + b = k \cdot q + c$. To check that c is within the correct range, we employ a lookup argument with the table $\boldsymbol{t} = (0, \dots, q-1)$. It is feasible in our design since the parameter q, the modulus used in the LWE scheme, is relatively small.

Definition 8. *The LWE addition relation $\mathcal{R}_{\mathsf{add}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (q, N, \boldsymbol{a}, \boldsymbol{b}), \boldsymbol{c})$$

where $a, b, c \in \mathbb{Z}_q^N$ and $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{c} \pmod q$.

Theorem 4. *For every finite field $\mathbb{F}$ and extension field $\mathbb{EF}$, and positive integers q, N, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{add}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $\frac{O(N)}{|\mathbb{EF}|}$ with $O(\log N)$ round complexity. The prover can be implemented in $O(N)$ field operations, and the verifier can be implemented in $O(\log N)$.*

We prove it with the following Construction **??**.

Construction 5. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{dec}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} =\perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, B, \ell)$, and witness $\mathbb{w} = (\boldsymbol{a}, (\boldsymbol{a}_0, \dots, \boldsymbol{a}_{\ell-1}))$.*

- *The prover $\mathcal{P}$ and verifier $\mathcal{V}$ sets $\boldsymbol{t} = (0, \dots, B-1, 0, \dots, 0) \in \mathbb{F}^N$ and run the batched lookup protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, N, \boldsymbol{t}), (\boldsymbol{a}_0, \dots, \boldsymbol{a}_{\ell-1}))$.*
- *The verifier samples a random point on $r \xleftarrow{\$} \mathbb{EF}^{\log N}$ and makes the queries for $\boldsymbol{a}$, $\boldsymbol{a}_0$, …, $\boldsymbol{a}_{\ell-1}$.*
  $$e^{\boldsymbol{a}} = \boldsymbol{a}(r), \quad e^{\boldsymbol{a}_0} = \boldsymbol{a}_0(r), \quad \dots, \quad e^{\boldsymbol{a}_{\ell-1}} = \boldsymbol{a}_{\ell-1}(r)$$
  check that $\sum_{i=0}^{\ell-1} B^i \cdot e^{\boldsymbol{a}_i} = e^{\boldsymbol{a}}$.

4.2. Batched Lift

In this section, we introduce PIOP for the lift operation before the accumulator updating. This operation lifts a ring element $a \in \mathbb{Z}_q$ into the exponent of a polynomial $X^b \in G$ where $G = \{X, \dots, X^{2N}\}$ is a cyclic subgroup of $\mathcal{R}_Q = \mathbb{F}/(X^N + 1)$ and $b = a \cdot \frac{2N}{q} \in \mathbb{Z}_{2N}$.

Firstly, we need to perform an additional modular operation on X^b with the modulus X^N+1 when $b \geq N$ because the polynomial in $\mathcal{R}_Q$ is uniquely represented with its coefficient vector $\boldsymbol{c} = (c_0, \dots, c_{N-1}) \in \mathbb{F}^N$. Given $b \in \mathbb{Z}_{2N}$, the modular result is a monomial $(1-2k) \cdot X^r \in \mathcal{R}_Q$, where $b = k \cdot N + r$ with $k \in \{0,1\}$ and $r \in [N]$. Specifically, the monomial is X^r if $b < N$ and $-X^r$ otherwise. We refer to this monomial as the lifted polynomial, which can be represented by a coefficient vector $\boldsymbol{c} \in \mathbb{F}^N$ with only one non-zero value $1-2k$, located at position r.

When we consider the batched lift operation for an input vector $\boldsymbol{b} = (b_0, \dots, b_{M-1}) \in \mathbb{Z}_{2N}^M$, all the lifted polynomials constitute a sparse coefficient matrix $C^{N\times M} = (\boldsymbol{c}_0, \dots, \boldsymbol{c}_{M-1})$ with each column containing exactly one non-zero element $1-2k_i$, located at position r_i, where $b_i = k_i \cdot N + r_i$ with $k_i \in \{0,1\}$ and $r_i \in [N]$.

A key observation is that all lifted polynomials are subsequently multiplied by another polynomial in the accumulator updating. Each multiplication involves an NTT operation on the lifted polynomial to obtain its evaluation vector, denoted by $\boldsymbol{d}_i = \mathsf{NTT}(\boldsymbol{c}_i)$ for $i \in [M]$, which can be proved by the PIOP for the NTT relation $\mathcal{R}_{\mathsf{NTT}}$, returning subclaims for the evaluations of the coefficient vectors $\tilde{\boldsymbol{c}}_0(u), \dots, \tilde{\boldsymbol{c}}_{M-1}(u)$ at the point $u \xleftarrow{\$} \mathbb{EF}^{\log N}$. These subclaims can be further reduced to the evaluation on the sparse coefficient matrix $\tilde{C}(u,v)$ by the random linear combination, where $v \xleftarrow{\$} \mathbb{EF}^{\log M}$. Let $\tilde{\boldsymbol{d}}(y)$ be the multilinear extension defined by the vector $\boldsymbol{d} = (\tilde{\boldsymbol{c}}_0(u), \dots, \tilde{\boldsymbol{c}}_{M-1}(u))$, it is easy to verify the following:

$$\begin{aligned}
\tilde{\boldsymbol{d}}(v) &= \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}(y, v)\, \tilde{\boldsymbol{c}}_{\mathcal{Y}}(u) \\
&= \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}(y, v) \sum_{x \in \{0,1\}^{\log N}} \tilde{eq}(x, u)\, \boldsymbol{c}_{\mathcal{Y}}(x) \\
&= \sum_{y \in \{0,1\}^{\log M}} \sum_{x \in \{0,1\}^{\log N}} \tilde{eq}(y, v)\, \tilde{eq}(x, u)\, C(\mathcal{X}, \mathcal{Y}) \\
&= \tilde{C}(u, v)
\end{aligned} \tag{5}$$

Hence, instead of explicitly proving the lift operation from the input vector $\boldsymbol{b} \in \mathbb{Z}_{2N}^M$ to the coefficient matrix $C^{N\times M}$, we directly prove the following M NTT operations, returning a batch of claims to the evaluations of $\tilde{\boldsymbol{c}}_0(u), \dots, \tilde{\boldsymbol{c}}_{M-1}(u)$. By random linear combination, these claims are reduced to the evaluation on the sparse matrix $\tilde{C}(u,v)$, which can be eventually reduced to the evaluation of $\mathsf{row} = \boldsymbol{r}$ and $\mathsf{val} = 1 - 2\boldsymbol{k}$ as discussed in Section 3.1, where $\boldsymbol{k}$ and $\boldsymbol{r}$ are both derived from the input vector $\boldsymbol{b}$.

Additionally, by applying the method proposed in Section 3.2 for proving batched NTT, the prover time is reduced to $O(M+N)$ due to the sparsity of the coefficient vectors.

Definition 9. *The batched lift relation $\mathcal{R}_{\mathsf{lift}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, \mathcal{R}_Q, N, M), (\boldsymbol{s}, \boldsymbol{a}_0, \dots, \boldsymbol{a}_{M-1})))$$

where $\boldsymbol{s} \in \mathbb{Z}_{2N}^M$, and for each $i \in [M]$, we have $\boldsymbol{a}_i = \mathsf{NTT}(\boldsymbol{c}_i)$ where $\boldsymbol{c}_i$ is the coefficient vector of the polynomial $X^{s_i} \mod X^N + 1$ in $\mathcal{R}_Q$.

Theorem 5. *For every finite field $\mathbb{F}$, extension field $\mathbb{EF}$ and the quotient ring $\mathcal{R}_Q$, and positive integers N, M, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{lift}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $O\left(\frac{\log N + M}{|\mathbb{EF}|}\right)$ with $O(\log N + \log M)$ round complexity. The* *prover can be implemented in $O(M+N)$ field operations, and the verifier can be implemented in $O(\log^2 N + M)$.*

We prove it with the following Construction 6.

Construction 6. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{lift}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, \mathcal{R}_Q, N, M)$, and witness $\mathbb{w} = (\boldsymbol{s}, \boldsymbol{a}_0, \dots, \boldsymbol{a}_{M-1})$.*

- *$\mathcal{P}$ computes $\boldsymbol{k}$ and $\boldsymbol{r}$ such that $\boldsymbol{s} = \boldsymbol{k} \cdot N + \boldsymbol{r}$ where $\boldsymbol{k} \in \{0,1\}$ and $\boldsymbol{r} \in \mathbb{Z}_N^N$ and sends them to the verifier.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\{\boldsymbol{k}\}, \{1-\boldsymbol{k}\}, 0))$.*
- *Define table $\boldsymbol{t} = (0, \dots, N-1)$, and $\mathcal{P}$ and $\mathcal{V}$ run the lookup protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}, 1), \boldsymbol{r})$*
- *The verifier samples a random point on $\tau \xleftarrow{\$} \mathbb{EF}^{\log N}$ and makes the queries for $\boldsymbol{a}_0, \dots, \boldsymbol{a}_{M-1}$:*
  $$e_0 = \boldsymbol{a}_0(\tau), \quad \dots, \quad e_{M-1} = \boldsymbol{a}_{M-1}(\tau)$$
- *$\mathcal{P}$ computes the $\boldsymbol{c} = \sum_{i=0}^{M-1} \rho_i \cdot \boldsymbol{c}_i$, where $\boldsymbol{c}_i = (1-2k_i) \cdot X^{r_i} \in \mathcal{R}_Q$ with only one non-zero entrie.*
  // Note that this computation can be done in $O(M)$. The evaluation on $\tilde{\boldsymbol{c}}$ can queried via $\tilde{\boldsymbol{c}}_0, \dots, \tilde{\boldsymbol{c}}_{M-1}$.
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the NTT protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, u, \sum_{i=0}^{M-1} \rho_i \cdot e_i), (\boldsymbol{c}))$ and return the subclaim $E \overset{?}{=} \tilde{\boldsymbol{c}}(u)$ at a random point u.*
- *The verifier $\mathcal{V}$ samples a challenge $\rho \xleftarrow{\$} \mathbb{EF}^M$ uniformly at random and sends it to the prover $\mathcal{P}$.*
- *For each $i \in [M]$, the prover computes $d_i = \tilde{\boldsymbol{c}}_i(u)$ and sends $\boldsymbol{d} = (d_0, \dots, d_{M-1}) \in \mathbb{EF}^M$ to the verifier.*
- *The verifier first checks $E = \sum_{i=0}^{M-1} \rho_i \cdot d_i$.*
  // Then, the prover and the verifier needs to check $d_i = \tilde{\boldsymbol{c}}_i(u)$ for every $i \in [M]$
- *The verifier $\mathcal{V}$ samples a challenge $v \xleftarrow{\$} \mathbb{EF}^{\log M}$ uniformly and sends it to the prover $\mathcal{P}$.*
- *Let $\psi = \sum_{x \in \{0,1\}^{\log M}} \tilde{eq}(v, x) \cdot \boldsymbol{d}(x)$ and $\mathsf{val} = 1 - 2\boldsymbol{k}$ be an oracle that can be queried via $\boldsymbol{k}$.*
- *The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run the sparse matrix evaluation protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, M, \psi, u, v), (\mathsf{val}, \boldsymbol{r}))$.*
- *The verifier $\mathcal{V}$ makes the evaluation query for $\boldsymbol{a}, \boldsymbol{k}, \boldsymbol{r}$:*
  $$e^{\boldsymbol{a}} := \boldsymbol{a}(\tau), \quad e^{\boldsymbol{k}} := \boldsymbol{k}(\tau), \quad e^{\boldsymbol{r}} := \boldsymbol{r}(\tau)$$
  check that $e^{\boldsymbol{a}} = e^{\boldsymbol{k}} \cdot N + e^{\boldsymbol{r}}$.

4.3. Accumulator Updating

In this section, we propose a PIOP for accumulator updating:
$$\mathsf{acc}' = \mathsf{acc} + \boldsymbol{d} \otimes \mathsf{acc} \diamond \mathsf{Z},$$
which updates the accumulator state from acc to acc'. Here $\mathsf{acc} \in \mathbb{RLWE}$ (and similarly for acc') consists of two vectors $\mathsf{acc}.\boldsymbol{a}$ and $\mathsf{acc}.\boldsymbol{b}$ in $\mathbb{F}^N$. Each update consumes a vector $\boldsymbol{d} \in \mathbb{F}^N$ and $\mathsf{Z} \in \mathbb{RGSW}$, which contains 4ℓ vectors $\mathsf{Z}.\boldsymbol{a}_0, \dots, \mathsf{Z}.\boldsymbol{a}_{2\ell-1}, \mathsf{Z}.\boldsymbol{b}_0, \dots, \mathsf{Z}.\boldsymbol{b}_{2\ell-1} \in \mathbb{F}^N$.

As defined in Section 2.2, the operation $\otimes$ takes as input a vector $\boldsymbol{d} \in \mathbb{F}^N$ and $(\mathsf{acc}.\boldsymbol{a}, \mathsf{acc}.\boldsymbol{b}) \in \mathbb{RLWE}$, and computes the intermedia result:
$$\mathsf{Imult} = (\mathsf{INTT}(\boldsymbol{d} \circ \mathsf{acc}.\boldsymbol{a}), \mathsf{INTT}(\boldsymbol{d} \circ \mathsf{acc}.\boldsymbol{b})),$$
where $\mathsf{Imult} \in \mathbb{RLWE}$. Thus, each $\otimes$ operation consists of two Hardmard products and 2 INTT operations.

Next, the operation $\diamond$ takes Imult as the input and performs two gadget decomposition $\mathsf{Dec}(\mathsf{Imult}.\boldsymbol{a}; B, \ell)$ and $\mathsf{Dec}(\mathsf{Imult}.\boldsymbol{b}; B, \ell)$, resulting 2ℓ gadgets. We flatten these decomposed gadgets into $\mathsf{bits}[0..2\ell] = (\mathsf{Dec}(\mathsf{Imult}.\boldsymbol{a}; B, \ell), \mathsf{Dec}(\mathsf{Imult}.\boldsymbol{b}; B, \ell))$, and compute $\mathsf{prod} =$
$$\left(\sum_{i=0}^{2\ell-1} \mathsf{NTT}(\mathsf{bits}[i]) \circ \mathsf{Z}.\boldsymbol{a}_i, \sum_{i=0}^{2\ell-1} \mathsf{NTT}(\mathsf{bits}[i]) \circ \mathsf{Z}.\boldsymbol{b}_i \right)$$
where $\mathsf{prod} \in \mathbb{RLWE}$. Hence, each $\diamond$ operation consists of 2ℓ NTT operations and two sums of Hadamard products.

To put it all together, each update involves $2\ell + 2$ NTT/INTT operations, two gadget decompositions, and four (sums of) Hadamard products. The PIOP for proving the accumulator update relation $\mathcal{R}_{\mathsf{acc}}$, defined as follows, can be easily derived from the aforementioned PIOPs.

Definition 10. *The accumulator updating relation $\mathcal{R}_{\mathsf{acc}}$ is the set of tuples*
$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\mathsf{Z}, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, B, \ell), (\mathsf{acc}, \boldsymbol{d}, \mathsf{acc}'))$$
where $\ell = \lceil \log_B Q \rceil$, $\mathsf{Z} \in \mathbb{RGSW}$, $\mathsf{acc}, \mathsf{acc}' \in \mathbb{RLWE}$, $d \in \mathbb{F}^N$, and $\mathsf{acc}' = \mathsf{acc} + \boldsymbol{d} \otimes \mathsf{acc} \diamond \mathsf{Z}$.

Theorem 6. *For every finite field $\mathbb{F}$, extension field $\mathbb{EF}$, and positive integers N, B, ℓ, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{acc}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $O\left(\frac{\ell N}{|\mathbb{EF}|}\right)$ with $O(\log^2 N)$ round complexity. The prover can be implemented in $O(\ell N)$ field operations, and the verifier can be implemented in $O(\log^2 N + \ell)$.*

We prove it with the following Construction 7

Construction 7. *We construct a PIOP for the accumulator update. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \mathsf{Z}$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, B, \ell)$, and witness $\mathbb{w} = (\mathsf{acc}, \boldsymbol{d}, \mathsf{acc}')$.*

- *$\mathcal{P}$ computes the following traces:*
  - $\mathsf{mult} = (\boldsymbol{d} \circ \mathsf{acc}.\boldsymbol{a}, \boldsymbol{d} \circ \mathsf{acc}.\boldsymbol{b}) \in \mathbb{RLWE}$
  - $\mathsf{Imult} = (\mathsf{INTT}(\mathsf{mult}.\boldsymbol{a}), \mathsf{INTT}(\mathsf{mult}.\boldsymbol{b})) \in \mathbb{RLWE}$
  - $\mathsf{bits}[0..\ell] = \mathsf{Dec}(\mathsf{Imult}.\boldsymbol{a}; B, \ell)$
  - $\mathsf{bits}[\ell..2\ell] = \mathsf{Dec}(\mathsf{Imult}.\boldsymbol{b}; B, \ell)$
  - $\mathsf{Nbits}[i] = \mathsf{NTT}(\mathsf{bits}[i])$ *for* $i \in [2\ell]$
  - $\mathsf{prod} = \left(\sum_{i=0}^{2\ell-1} \mathsf{Nbits}[i] \circ \mathsf{Z}.\boldsymbol{a}_i, \sum_{i=0}^{2\ell-1} \mathsf{Nbits}[i] \circ \mathsf{Z}.\boldsymbol{b}_i\right)$
- *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) =$*
  - $(\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{d}, \mathsf{acc}.\boldsymbol{a}, \mathsf{mult}.\boldsymbol{a}))$
  - $(\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{d}, \mathsf{acc}.\boldsymbol{b}, \mathsf{mult}.\boldsymbol{b}))$
  - $(\perp, (\mathbb{F}, \mathbb{EF}, N, 2\ell), (\mathsf{Nbits}[0..2\ell], \mathsf{Z}.\vec{\boldsymbol{a}}, \mathsf{prod}.\boldsymbol{a}))$
  - $(\perp, (\mathbb{F}, \mathbb{EF}, N, 2\ell), (\mathsf{Nbits}[0..2\ell], \mathsf{Z}.\vec{\boldsymbol{b}}, \mathsf{prod}.\boldsymbol{b}))$
- *$\mathcal{P}$ and $\mathcal{V}$ run the gadget decomposition protocol with*
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell), (\mathsf{Imult}.\boldsymbol{a}, \mathsf{bits}[0..\ell]))$
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell), (\mathsf{Imult}.\boldsymbol{b}, \mathsf{bits}[\ell..2\ell]))$
- *$\mathcal{V}$ samples a random point $r \in \mathbb{F}^{\log N}$ and make queries for $e^{\mathsf{mult}.\boldsymbol{a}} = \mathsf{mult}.\boldsymbol{a}(r)$, $e^{\mathsf{mult}.\boldsymbol{b}} = \mathsf{mult}.\boldsymbol{b}(r)$, and $e_i = \mathsf{Nbits}[i](r)$ for each $i \in [2\ell]$ and also check that*

$e^{\mathsf{acc}'.\boldsymbol{a}} = e^{\mathsf{acc}.\boldsymbol{a}} + e^{\mathsf{prod}.\boldsymbol{a}}$, $e^{\mathsf{acc}'.\boldsymbol{b}} = e^{\mathsf{acc}.\boldsymbol{b}} + e^{\mathsf{prod}.\boldsymbol{b}}$ *with corresponding queries.*

- *The prover and the verifier run the NTT protocol with*
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, e^{\mathsf{mult}.\boldsymbol{a}}), (\mathsf{Imult}.\boldsymbol{a}))$
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, e^{\mathsf{mult}.\boldsymbol{b}}), (\mathsf{Imult}.\boldsymbol{b}))$
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, e_i), (\mathsf{bits}[i]))$ *for* $i \in [2\ell]$

4.4. Modulus Switching

The modulus switching maps a field element $a \in \mathbb{F}$ with modulus Q to a ring element $b \in \mathbb{Z}_q$ with modulus q. Specifically, the modulus switching function is defined as follows:

$$b = \lfloor \frac{a \cdot q}{Q} \rceil \pmod q$$

which rounds the fraction $\frac{a \cdot q}{Q}$ to the nearest integer, followed by a modular operation with modulus q. Note that $\lfloor \frac{a \cdot q}{Q} \rceil$ may wrap around the modulus q, resulting in $b = 0$.

In our work, we consider modulus switching from Q to q under the assumption[2] of $2q|Q-1$. Define $k = \frac{Q-1}{2q}$, which gives the equality $Q - 1 = 2kq$.

A key observation is that the modulus switching with such equality can be considered as a structured mapping, as illustrated in Fig. 3. This mapping takes a sequential range in $\mathbb{F}$ and maps it to the same element in $\mathbb{Z}_q$.

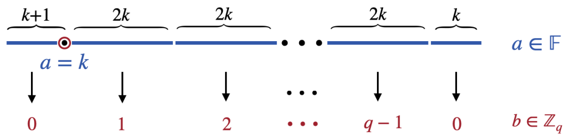


Figure 3: Assuming $2q|Q-1$ with $k = \frac{Q-1}{2q}$, the modulus switching maps $a \in [0, k] \cup [Q-k, Q)$ to 0, and $a \in [(2b-1) \cdot k + 1, (2b+1) \cdot k]$ for each $b \in \{1, \ldots, q-1\}$ to $b \in \mathbb{Z}_q$.

To handle the special range $[0, k] \cup [Q-k, Q)$ shown in Fig. 3, we express it as $k \cup [(2q-1) \cdot k + 1, (2q+1) \cdot k] \pmod Q$ based on the equality $Q - 1 = 2kq$. By introducing an additional witness $b' \in \{1, \ldots, q\}$ such that $b \equiv b' \mod q$, we can prove the modulus switching relation $\mathcal{R}_{\mathsf{switch}}$ (as defined in Definition 11) based on the dichotomy presented in Theorem 7.

Theorem 7. *Let q be a power of two, and let Q be a prime such that $2q|Q-1$. Define $k = \frac{Q-1}{2q}$, so that $Q = 2kq + 1$. For any $a \in \mathbb{F}_Q$ and $b \in \mathbb{Z}_q$, we have $b = \lfloor \frac{a \cdot q}{Q} \rceil$ if and only if either $a = k$ and $b = 0$, or there exists $b' \in \{1, \ldots, q\}$ such that $b' \equiv b \mod q$ and $a \in [(2b'-1) \cdot k+1, (2b'+1) \cdot k] \in \mathbb{F}$.*

Proof. First, it is trivial to observe that $a = k \notin [(2b'-1) \cdot k + 1, (2b'+1) \cdot k]$ for every $b' \in \{1, \ldots, q\}$. In this case, if $a = k$, we have $\lfloor \frac{k \cdot q}{Q} \rceil = 0$ since $Q - 1 = 2kq$.

[2]This assumption is easily satisfied in the FHEW/TFHE schemes, which requires $2N|Q-1$ and $q|2N$. The prime Q, with length ℓ, is sampled as $Q = 2^\ell - i \cdot 2N + 1$ for some integer i, implying $q|Q-1$, assuming q is a power of two in our work.

Now, consider $a \in \mathbb{F} \backslash \{k\}$, we can rewrite the equality $kq = \frac{1}{2}(Q-1)$. For any $a \in [(2b'-1) \cdot k+1, (2b'+1) \cdot k] \subseteq \mathbb{F}$, where $b' \in \{1, \ldots, q\}$, the lower bound is given by:

$$a \cdot q \geq ((2b'-1) \cdot k+1) \cdot q = (2b'-1) \cdot \frac{Q-1}{2} + q > b'Q - \frac{1}{2}Q,$$

where the last inequality holds because $2q + 1 - 2b' > 0$ for any $b' \in \{1, \ldots, q\}$.

Next, we compute the upper bound, which we handle in two cases:

Case 1: For any $a \in [(2b'-1) \cdot k + 1, (2b'+1) \cdot k] \subseteq \mathbb{F}$ where $b' = b \in \{1, \ldots, q-1\}$, we have

$$a \cdot q \leq (2b'+1) \cdot kq = (b' + \frac{1}{2}) \cdot (Q-1) < b'Q + \frac{1}{2}Q$$

Thus, we conclude that $\lfloor \frac{a \cdot q}{Q} \rceil = b' = b$.

Case 2: For any $a \in [0, k) \cup [Q-k, Q)$, which corresponds to $a \in [(2b'-1) \cdot k + 1, (2b'+1) \cdot k] \pmod Q$ with $b' = q$ and $b = 0$, we consider the following two subcases:

1. If $a \in [0, k)$, we have $a \cdot q < kq = \frac{1}{2}(Q-1) < \frac{1}{2}Q$.
2. If $a \in [Q-k, Q)$, we have $a \cdot q < q \cdot Q$.

Combining these two subcases, we conclude that $\lfloor \frac{a \cdot q}{Q} \rceil = 0 = b \mod q$.

Finally, we conclude that $\lfloor \frac{a \cdot q}{Q} \rceil = b$. The converse direction can be proven similarly, and we omit the details here. $\square$

Remark 4. *Lemma 2 in Appendix A.1 extends to modulus switching from Q to q with $q|Q-1$, by replacing the rounding operation with the floor operation, i.e. $\lfloor \frac{a \cdot q}{Q} \rfloor \in \mathbb{Z}_q$.*

Hence, the relation between $a \in \mathbb{F}_Q$ and $b \in \mathbb{Z}_q$ defined by modulus switching operation is reduced to a dichotomy, where each case can be expressed as an equality. If the first case holds, i.e., $a = k$ and $b = 0$, we use a random linear combination to prove that $p = \lambda_1 \cdot (a - k) + \lambda_2 \cdot b = 0$, where $\lambda_1 \overset{\$}{\leftarrow} \mathbb{EF}$ and $\lambda_2 \overset{\$}{\leftarrow} \mathbb{EF}$. Otherwise, we prove that $c = a - (2b'-1) \cdot k - 1 \in \mathbb{F}$ lies within the range $[0, 2k)$ using the range check protocol derived from gadget decomposition method in Section 3.3. Combining the two equalities with an additional witness $w \in \{0, 1\}$, which indicates the dichotomy, we obtain the following equality:

$$w \cdot p + (1 - w) \cdot (a - (2b'-1) \cdot k - 1 - c)$$

where the witness $w = \mathbb{1}(a = k)$ can be easily computed. Based on this idea, we propose a PIOP for the modulus switching relation $\mathcal{R}_{\mathsf{switch}}$, which involves the sum of Hadamard products and range checks.

Definition 11. *The modulus switching relation $\mathcal{R}_{\mathsf{switch}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, Q, q, k, B, \ell, \boldsymbol{b}), \boldsymbol{a})$$

where Q is the prime of $\mathbb{F}$, $2q \mid Q-1$, $k = \frac{Q-1}{2q}$, $\ell = \lceil \log_B(2k) \rceil$, $\boldsymbol{a} \in \mathbb{F}^N$, $\boldsymbol{b} \in \mathbb{Z}_q^N$, and for each $i \in [N]$, we have $b_i = \lfloor \frac{a_i \cdot q}{Q} \rceil$.

Theorem 8. *For every finite field $\mathbb{F}$, extension field $\mathbb{EF}$, and positive integers N, q, k, B, there is a PIOP for the indexed* *relation $\mathcal{R}_{\mathsf{switch}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $O\left(\frac{\ell N}{|\mathbb{EF}|}\right)$ with $O(\log N)$ round complexity. The prover can be implemented in $O(\ell N)$ field operations, and the verifier can be implemented in $O(\log N + \ell)$.*

We prove it with the following Construction 8.

Construction 8. *We construct a PIOP for the indexed relation $\mathcal{R}_{\mathsf{switch}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, Q, q, k, B, \ell, \boldsymbol{b})$, and witness $\mathbb{w} = \boldsymbol{a}$.*

- *$\mathcal{P}$ computes $\mathsf{id_a}, \mathsf{id_b}, \boldsymbol{b}'$ and $\boldsymbol{c}$ for each $x \in \{0,1\}^{\log N}$:*
  - $\mathsf{id_a}(x) = \mathbb{1}(\boldsymbol{a}(x) = k)$
  - $\mathsf{id_b}(x) = \mathbb{1}(\boldsymbol{b}(x) = 0)$
  - $\boldsymbol{b}'(x) = \mathsf{id_b}(x) \cdot q + \boldsymbol{b}(x)$
  - $\boldsymbol{c}(x) = (1 - \mathsf{id_a}(x)) \cdot (\boldsymbol{a}(x) - (2\boldsymbol{b}'(x) - 1) \cdot k - 1)$

  and sends $\mathsf{id_a}, \mathsf{id_b}$ and $\boldsymbol{c}$ to the verifier $\mathcal{V}$, and $\boldsymbol{b}'$ can be queried via $\boldsymbol{b}$ and $\mathsf{id_b}$.
- *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with*
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1)(\{\mathsf{id_a}\}, \{1 - \mathsf{id_a}\}, \boldsymbol{0}))$
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1)(\{\mathsf{id_b}\}, \{1 - \mathsf{id_b}\}, \boldsymbol{0}))$
- *$\mathcal{P}$ and $\mathcal{V}$ run the range check protocol on $[0, 2k)$ with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell, 2k), \boldsymbol{c})$*
- *The verifier $\mathcal{V}$ samples challenges $\lambda_1 \xleftarrow{\$} \mathbb{EF}$ and $\lambda_2 \xleftarrow{\$} \mathbb{EF}$ uniformly at random and sends it to the prover $\mathcal{P}$.*
- *Let $\mathsf{id_{point}} = \lambda_1 \cdot (\boldsymbol{a} - k) + \lambda_2 \cdot \boldsymbol{b}$ be an oracle that can be queried via $\boldsymbol{a}$ and $\boldsymbol{b}$.*
- *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 2), (\{\mathsf{id_a}, 1 - \mathsf{id_a}\}, \{\mathsf{id_{point}}, (\boldsymbol{a} - (2\boldsymbol{b}' - 1) \cdot k - 1 - \boldsymbol{c})\}, \boldsymbol{0}))$.*

4.5. PIOP for FHE NAND

We propose our PIOP for the FHE NAND circuit described in Algorithm 2, corresponding to the process illustrated in Fig. 1. By invoking the PIOPs for all the relations defined for corresponding operations in the algorithm, we can propose the PIOP for the NAND relation $\mathcal{R}_{\mathsf{NAND}}$.

As shown in the Algorithm 2, it takes as input two $\mathbb{LWE}$ ciphertexts, $\mathsf{ct}_0, \mathsf{ct}_1$, a constant vector $\boldsymbol{v}$, and n $\mathbb{RGSW}$ ciphertexts $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1}$, and produces as outputs the desired $\mathbb{LWE}$ ciphertext. The computation trace contains n $\mathbb{RGSW}$ values $\{\mathsf{acc}[i] \in \mathbb{RGSW}\}$, storing the accumulator states in each update. The entire process involves one LWE addition, $n+1$ lift operations, a Hadamard product, n accumulator updates, and $n+1$ modulus switching operations, along with some linear operations.

Definition 12. *The NAND relation $\mathcal{R}_{\mathsf{NAND}}$ is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) =$*

$$((\boldsymbol{v}, \{\mathsf{Z}_i\}_{i=0}^{n-1}), (\mathbb{F}, \mathbb{EF}, N, n, q, B, \mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}), \perp)$$

where $\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct} \in \mathbb{LWE}$, $\boldsymbol{v} \in \mathbb{F}^N$, $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1} \in \mathbb{RGSW}$ and $\mathsf{ct} = \mathsf{HomNAND}(\mathsf{ct}_0, \mathsf{ct}_1, \boldsymbol{v}, \mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1})$.

Theorem 9. *For every finite field $\mathbb{F}$, extension field $\mathbb{EF}$, and positive integers N, q, k, B, there is a PIOP for the indexed relation $\mathcal{R}_{\mathsf{NAND}}$ that supports instances over $\mathbb{F}$, with perfect completeness and soundness error $O\left(\frac{\ell N n}{|\mathbb{EF}|}\right)$ with $O(\log n + \log^2 N)$ round complexity where $\ell = \lceil \log_B Q \rceil$. The prover can be implemented in $O(\ell N n)$ field operations, and the verifier can be implemented in $O(\log^2 N + \ell n)$.*

Algorithm 2 $\mathsf{HomNAND}(\mathsf{ct}_0, \mathsf{ct}_1, \boldsymbol{v}, \mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1})$

```
Input: $\mathsf{ct}_0, \mathsf{ct}_1 \in \mathbb{LWE}$
Input: $\boldsymbol{v} \in \mathbb{F}^N$ and $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1} \in \mathbb{RGSW}$
Output: $\mathsf{ct} \in \mathbb{LWE}$
 1: $\boldsymbol{a} = \mathsf{ct}_0 + \mathsf{ct}_1 \pmod q$    // LWE Addition
 2: for $i = 0$ to $n-1$ do
 3:     $\boldsymbol{b}_i = q - a_i$
 4: end for
 5: $b_n = a_n$
 6: $\boldsymbol{c} = (\boldsymbol{b} \cdot \frac{2N}{q}) \in \mathbb{Z}_{2N}^{n+1}$
 7: for $i = 0$ to $n$ do // Batched Lift
 8:     $\boldsymbol{d}_i = \mathsf{NTT}(X^{c_i} \mod X^N + 1)$
 9: end for
10: $\mathsf{acc}[0] = (\boldsymbol{0}, \boldsymbol{v} \circ \boldsymbol{d}_n) \in \mathbb{RLWE}$    // Hadamard Product
11: for $i = 0$ to $n-1$ do    // Accumulator Update
12:     $\mathsf{acc}[i+1] = \mathsf{acc}[i] + (\boldsymbol{d}_i - 1) \otimes \mathsf{acc}[i] \diamond \mathsf{Z}_i$
13: end for
14: for $i = 0$ to $n-1$ do    // Modulus Switching
15:     $a'_i = \lfloor \frac{(\mathsf{acc}[n].a_i) \cdot q}{Q} \rceil$
16: end for
17: $b' = \lfloor \frac{(\mathsf{acc}[n].b_0 + 8/Q) \cdot q}{Q} \rceil$
18: return $\mathsf{ct} = (\boldsymbol{a}', b')$
```

Construction 9. *We construct a PIOP for the $\mathcal{R}_{\mathsf{NAND}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = (\boldsymbol{v}, \{\mathsf{Z}_i\}_{i=0}^{n-1})$, instance $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, n, q, B, \mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct})$, and witness $\mathbb{w} = \perp$.*

- *The prover invokes the Algorithm 2, storing the vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, the NTT results of the lifted polynomials $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_n$, and the accumulator states, including $\mathsf{acc}[0].\boldsymbol{b}$ and $\{\mathsf{acc}[i]\}_{i=1}^{n}$. These values (except for $\boldsymbol{c}$) are sent to the verifier as oracle messages, while $\boldsymbol{c}$ is an oracle that can be queried via $\boldsymbol{a}$.*
- *The prover and the verifier run the LWE-Addition protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (q, n+1, \mathsf{ct}_0, \mathsf{ct}_1), \boldsymbol{a})$.*
- *Let $\mathsf{ext_b} = (0, \ldots, 0, 1) \in \mathbb{Z}_2^{n+1}$.*
- *The prover and the verifier run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 2), (\{q - \boldsymbol{a}, \boldsymbol{a}\}, \{1 - \mathsf{ext_b}, \mathsf{ext_b}\}, \boldsymbol{b}))$.*
- *The prover and the verifier run the Lift protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, \mathcal{R}_Q, N, n+1), (\boldsymbol{c}, \boldsymbol{d}_0, \ldots, \boldsymbol{d}_n))$*
- *The prover and the verifier run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{v}, \boldsymbol{d}_n, \mathsf{acc}[0].\boldsymbol{b}))$*
- *For $i \in [n]$: the prover and the verifier run the accumulator updating protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) =$*
  - $(\mathsf{Z}, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, B, \ell), (\mathsf{acc}[i], \boldsymbol{d}_i - 1, \mathsf{acc}[i+1]))$
- *Let $\mathsf{ext_{b'}} = (1, 0, \ldots, 0) \in \mathbb{Z}_2^N$.*
- *The prover and the verifier run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\{\mathsf{acc}[n].\boldsymbol{b} + Q/8\}, \{\mathsf{ext_{b'}}\}, \boldsymbol{b}_{\mathsf{zero}}))$*

- *Let* $\mathsf{ct_a} = \mathsf{ct}[0..n], \mathsf{ct_b} = (\mathsf{ct}[n], 0, \ldots, 0) \in \mathbb{Z}_q^n$ *be derived from the public input* ct.
- *The prover and the verifier run the Modulus Switching protocol with*
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, Q, q, k, B, \ell, \mathsf{ct}.\boldsymbol{a}), \boldsymbol{a}')$
  - $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, Q, q, k, B, \ell, \mathsf{ct}.\boldsymbol{b}), \boldsymbol{b}')$

Complexity Analysis. The heaviest part of the algorithm is the accumulator updating, which involves $n \cdot (2\ell + 2)$ NTT/INTT operations, $2n$ gadget decompositions with $2n\ell$ decomposed gadgets, and $4n$ sums of Hadamard products. The gadget decompositions are reduced to a batched lookup protocol with $2n\ell$ vectors, each of size N, using the table $\boldsymbol{t} = (0, \ldots, B-1)$. As a result, the soundness error and the prover time are both bound by the term $O(\ell N n)$. Since all these PIOPs are powered by sumcheck protocols, all of them, except for the delegation part of the NTT protocol, can be combined into a single grand sumcheck protocol. Consequently, the verifier time is determined by the rounds complexity of the NTT protocol, plus the query complexity, corresponding to the number of all multilinear extensions (vectors). The asymptotic complexity of each PIOP is summarized in Table 1.

5. HasteBoots: SNARGs for FHE NAND

Most modern SNARGs work by combining a type of interactive protocol known as polynomial IOP (PIOP) with a cryptographic primitive called a polynomial commitment scheme (PCS). This combination yields a succinct interactive argument, which can then be rendered non-interactive via the Fiat-Shamir transformation[39], yielding a SNARG. By using the compiler in [10] to combine the PIOP for FHE NAND in Section 4.5 and the Brakedown PCS[16], we build HasteBoots, a SNARG system that can efficiently generate proofs of correct homomorphic NAND computation.

5.1. Definitions

In our setting, given the bootstrapping key $\{\mathsf{Z}_i\}_{i=0}^{n-1}$ and the constant vector $\boldsymbol{v}$, the prover and the verifier both hold two public input $\mathbb{LWE}$ ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$ as well as the output $\mathbb{LWE}$ ciphertext ct. It is important to note in this setting, the FHE circuit to be evaluated under $\mathbb{LWE}$ ciphertexts is public. As a result, the prover does not have any private data to protect with, meaning the witness is empty. Given this setup, a SNARG system is sufficient without requiring knowledge extractors (i.e., SNARKs) nor requiring zero-knowledge.

Formally speaking, let the index $\mathbb{i}$ contain the bootstrapping key and the constant vector, and the public input $\mathbb{x} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct})$ be the $\mathbb{LWE}$ ciphertexts. The witness is empty, i.e. $\mathbb{w} = \perp$. HasteBoots is a tuple of algorithms (PPGen, KeyGen, Prove, Verify):

- $\mathsf{pp} \leftarrow \text{PPGen}(1^\lambda)$: Given the security parameter λ, this algorithm generates the public parameters pp.
- $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \text{KeyGen}(\mathbb{i}, \mathsf{pp})$: Given the index $\mathbb{i}$, this algorithm generates an index proving key ipk and the index verification key ivk.
- $\pi \leftarrow \text{Prove}(\mathbb{x}, \mathsf{ipk})$: Given the public input $\mathbb{x}$ and the proving key ipk, the algoritm generates the proof π.
- $\{0,1\} \leftarrow \text{Verify}(\mathbb{x}, \mathsf{ivk}, \pi)$: Given the public input $\mathbb{x}$ and the verifying key ivk, the algoritm verify the proof π.

We say that HasteBoots is a SNARG for the indexed relation $\mathcal{R}_{\mathsf{NAND}}$ if the following properties hold.

- **Completeness.** For any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{NAND}}$, $\mathsf{pp} \leftarrow \text{PPGen}(1^\lambda)$, $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \text{KeyGen}(\mathbb{i}, \mathsf{pp})$, and $\pi \leftarrow \text{Prove}(\mathbb{x}, \mathsf{ipk})$, we have $\Pr[\text{Verify}(\mathbb{x}, \mathsf{ivk}, \pi) = 1] = 1$.
- **Soundness.** For any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \text{PPGen}\left(1^\lambda\right) \\ (\mathsf{ipk}, \mathsf{ivp}) \leftarrow \text{KeyGen}(\mathbb{i}) \\ (\mathbb{x}, \pi^*) \leftarrow \mathcal{A}\left(1^\lambda, \mathsf{pp}, \mathsf{st}\right) \\ \text{Verify}(\mathbb{x}, \mathsf{ivk}, \pi^*, \mathsf{pp}) = 1 \end{array}\right].$$

5.2. Polynomial Commitment Scheme

Polynomial commitment schemes (PCS) are cryptographic protocols that allow the prover to commit to a polynomial and later proves its evaluations on the points queried by the verifier. PCS are widely used for realizing oracles in PIOP. In this paper, we focus on the multilinear polynomial commitment scheme. One of the prevailing multilinear polynomial commitment schemes is Brakedown PCS [16], featuring a linear time prover, which is both theoretically optimal and practically efficient.

For the security definition and formal constructions, see [16]. We describe our optimization of packing small polynomials, as FHE operations involve many small polynomials.

Packing Small Polynomials. The standard polynomial commitment schemes provide functionality to commit and query a single polynomial. In HasteBoots, we often need to query multiple polynomials at the same points. While pairing-based polynomial commitment schemes have group elements as their commitments, whose homomorphism improves the efficiency by batching the commitments and the queries, the hash-based polynomial commitment schemes such as Brakedown have hash value as their commitments, do not have significant advantages by batching. However, we observe that Brakedown PCS has poor performance for small polynomials; thus, we consider packing multiple small polynomials into a large one to improve efficiency.

Considering M small multilinear polynomials $\boldsymbol{c}_0(y), \ldots, \boldsymbol{c}_{M-1}(y)$, where $y \in \{0,1\}^{\log N}$, we can pack these polynomials into a large multilinear polynomial $C(x,y)$ such that for each $\mathcal{X} \in [M]$, it holds that $C(x,y) = \boldsymbol{c}_{\mathcal{X}}(y)$ for every $y \in \{0,1\}^{\log N}$. This large polynomial can be viewed as an $N \times M$ matrix, where the $\mathcal{Y}$-th column corresponds to the small multilinear polynomial $\boldsymbol{c}_{\mathcal{Y}}$. Consequently, the prover can only commit a single $(\log N + \log M)$-variate multilinear polynomial $C(x,y)$. When the verifier wants to query the values of $\tilde{\boldsymbol{c}}_0(v), \ldots, \tilde{\boldsymbol{c}}_{M-1}(v)$ at the same point $v \in \mathbb{EF}^{\log N}$, the prover first computes $d_i = \tilde{\boldsymbol{c}}_i(v)$ for each $i \in [M]$ and sends these values to the verifier. These subclaims can be reduced to the evaluations on the matrix $\tilde{C}(u,v)$, where

$u \xleftarrow{\$} \mathbb{EF}^{\log M}$ is sampled by the verifier. It is sufficient for the verifier to query the evaluation $\tilde{C}(u, v)$ at an extended point. The proof of correctness follows the same idea in Equation 5 when proving the reduction in the batched lift protocol in Section 4.2. This packing introduces soundness error of $\log M/|\mathbb{EF}|$ by the Schwartz-Zipple Lemma.

6. Implementation and Evaluation

We implemented HasteBoots as a library in about 33K lines of Rust code. Our library contains PIOPs for each component of the FHE circuit, an optimized PCS based on Brakedown [16], and a SNARG tailored for FHEW-like schemes. We plan to open-source our library in the future.

Our goal is to determine whether HasteBoots can prove FHE in practice, and thus answer the questions:

- Can HasteBoots generate proofs in seconds?
- What is the main bottleneck in the proof generation?

Experiment Setup. We select FHE parameters to optimize compatibility with our SNARG construction, setting $n = 1024$, $q = 1024$, $N = 1024$, and $Q = 2^{31} - 2^{27} + 1$. Our protocols operate in a 128-bit extension field of size Q^D with $D = 4$ to ensure the soundness while the FHE instance is defined over a base field of size Q.

Remark 5. *The parameters in our work are not exactly the same as those in [26], but they are quite similar, as many parameters in FHE need to be tweaked and optimized to ensure both correctness and security. In [26] they choose a parameter setting with $n = 728$, with Q being a 64-bit prime while their implementation also uses a 128-bit extension field of size Q^2. It is important to note that the circuit size grows with n, which means that we are effectively proving the correctness of a larger circuit compared to the one in their implementation.*

For gadget decomposition, we choose a gadget size of $B = 2^7$, which balances efficiency with the computational demands of FHE operations. Additionally, we implement the lookup protocol using LogUp [37], setting a batch block size of 3 to minimize overhead. This combination of parameters ensures that our approach remains both efficient and aligned with the underlying structure of FHE.

For Brakedown PCS [16], we set $\alpha = 0.1195$, $\beta = 0.0248$, and $r = 1.9$. Here, α controls the size of the subcode in each recursive call, β defines the code distance, and r adjusts the code rate. Additionally, we set a recursion stopping threshold of 10 to optimize performance.

Complexity Analysis. We summarize the complexity of HasteBoots's PIOP in Table 1. In our implementations, $M_1 = 4L_1n + 2n = 22528$, $M_2 = 2L_1n = 10240$, $M_3 = n + 1 = 1025$, $M_4 = n + 1 = 1025$ and there will be approximately $n(2L1 + 2) = 12288$ NTT instances. The complexity of the PCS can be found in [16], which results in $O(N)$ proving time, and $O(\sqrt{N})$ verification time and proof size, where N is the size of the polynomial.

End-to-End Performance. We first test the end-to-end performance of HasteBoots on an Apple M1, measuring the time spent on each component as detailed in Table 2. Note that we measure both PIOP and PCS time for each component, as their oracles are queried at different points and the opening proofs cannot be batched or packed. As shown, bootstrapping is the most time-consuming part of the entire process and remains the core challenge in the field. Within bootstrapping, accumulator updating accounts for over 90% of the time due to its complex operations, including NTT, gadget decomposition, and external product computations. Overall, HasteBoots achieves efficient proving times of approximately 7.6 seconds and verification times around 330 ms, with a proof size of approximately 156 MB. The verification time and proof size are mainly impacted by the Brakedown-based PCS, which, while fast in proof generation, inherently results in longer verification times and larger proof sizes. In particular, The prover spends around 15% of its time on the PCS, whereas PCS takes up 95% of the verifier's time. Future improvements could focus on exploring alternative PCS or recursive proofs [40, 41] to reduce the verification time and proof size.

Comparison with Prior Work. We compared the performance of HasteBoots with previous work on proving the bootstrapping process in FHE across different settings as shown in Table 3. The zkVM-based approaches, including those built on RISC0 [24] and SP1 [25], require approximately 3 days and 1 day, respectively, due to performance losses during the compilation of FHE programs. [26] leveraging Plonk [14] with custom gates to directly prove FHE, achieves improved efficiency with proof times around half an hour. However, as a general-purpose SNARK, Plonk still incurs some efficiency loss when handling FHE-specific operations. In contrast, HasteBoots is specifically designed for FHE, reducing the proof generation time to under 10 seconds. However, HasteBoots's use of linear code-based PCS results in slower verification times and larger proof sizes compared to [26], which constructs incremental verifiable computation (IVC) to prove accumulator updates with recursion in plonky2. This approach achieves a verification time of approximately 8 ms and a proof size of around 200 KB, whereas HasteBoots requires roughly 300 ms and 150 MB. It is worth noting that our proof size and verification time can be further optimized through recursion and by replacing the current Brakedown with more efficient PCS.

7. Related Works

Fully Homomorphic Encryption. There are two main categories of fully homomorphic encryption schemes. The first category is CKKS[2], BFV[3, 4], and BGV[5], which support the package of multiple messages into a single ciphertext, thus supporting SIMD-like batching homomorphic operations. However, their bootstrapping operations are very heavy. Although these schemes support leveled FHE without bootstrapping, they only support a small class of circuits with a maximum depth smaller than a certain upper bound and are not universal. The second category is FHEW[6], TFHE[7], and variants[42, 43, 44, 45]. They don't support batching, but they can perform very fast bootstrapping. We

Protocols	**Proving Time**	**Verification Time**	**Soundness** ($/\lvert\mathbb{EF}\rvert$)	**Rounds**	**Proof Size**	**Query**
Hadamard Protocol	$O(M_1N)$	$O(\log N + M_1)$	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(M_1)$
Batched Lookups	$O(M_2N)$	$O(\log N + M_2)$	$O(M_2N)$	$O(\log N)$	$O(\log N)$	$O(M_2)$
Sparse Matrix Evaluation	$O(M_3)$	$O(\log M_3)$	$O(M_3)$	$O(\log M_3)$	$O(\log M_3)$	$O(1)$
NTT	$O(N)$	$O(\log^2 N)$	$O(\log N)$	$O(\log^2 N)$	$O(\log^2 N)$	$O(1)$
Batched Lift	$O(M_4 + N)$	$O(\log^2 N + M_4)$	$O(\log N + M_4)$	$O(\log M_4 + \log^2 N)$	$O(\log^2 N + M_4)$	$O(M_4)$
Gadget Decomposition	$O(L_1N)$	$O(\log N + L_1)$	$O(L_1N)$	$O(\log N)$	$O(\log N)$	$O(L_1)$
Modulus Switching	$O(L_2N)$	$O(\log N + L_2)$	$O(L_2N)$	$O(\log N)$	$O(\log N)$	$O(L_2)$
LWE Addition	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$
Accumulator Updating	$O(L_1N)$	$O(\log^2 N + L_1)$	$O(L_1N)$	$O(\log^2 N)$	$O(\log^2 N)$	$O(L_1)$
FHE NAND	$O(L_1Nn)$	$O(\log^2 N + L_1n)$	$O(L_1Nn)$	$O(\log n + \log^2 N)$	$O(\log^2 N + n)$	$O(L_1n)$

TABLE 1: Complexity of PIOPs. **Notation:** B is the base size, q is a power of two. Define $L_1 = \lceil \log_B Q \rceil$ and $L_2 = \lceil \log_B(\frac{Q-1}{q}) \rceil$. Other parameters include N, the size of most vectors (corresponding to polynomial length in $\mathcal{R}_Q$), n, the dimension of the $\mathbb{LWE}$ vectors. Additionally, M_1 represents the number of Hadamard products within the num, M_2 the number of lookup vectors, M_3 the column size of the sparse matrix (also its sparsity), and M_4 the number of lift operations.

Operation		**Proving Time**	**Verification Time**
LWE Addition		19 ms	6 ms
Bootstrapping	Lift-NTT	160 ms	62 ms
	Acc Updating	7.4 s	262 ms
	Modulus Switching	31 ms	9 ms

TABLE 2: End-to-End performance for each components on Apple M1, which contains PIOP and PCS costs. Lift-NTT indicates the combined performance of the lift followed by the batched NTT, as we do not prove individual lift.

Proving Time	**RISC0** [24]	**SP1** [25]	[26]	**HasteBoots**
M3 Pro (8 cores)	–	–	40 min	**7 s**
C61.meta (128 cores)	–	–	21 min	**5 s**
Hpc7a.96xlarge (192 cores)	4600 min	1500 min	18 min	**4 s**
M4 Pro	–	–	–	**3 s**

TABLE 3: Proving time for a single bootstrapping operation across various devices.

focused on the FHEW scheme with GINX[46] bootstrapping method, since its bootstrapping is lightweight for SNARGs.

While FHE offers strong privacy, it lacks inherent guarantees for integrity. This has led to a line of research [47, 48, 49, 50, 51, 52] on attack that exploit vulnerabilities in ciphertext manipulation, potentially enabling adversaries to alter encrypted data and even recover the client's secret key.

Succinct Arguments. It has been extensively studied to provide efficient, verifiable proofs for computational integrity. Existing SNARKs [8, 9, 10, 11, 12, 13, 14, 15, 16] are designed for arbitrary computations. These works express computations in constraint systems like Rank-1 Constraint Systems (R1CS) or arithmetic circuits. Zero-knowledge Virtual Machine (zkVM) [24, 25] build on general-purpose SNARKs to compile high-level language programs into proofs. However, neither general-purpose SNARKs nor zkVMs account for the specific arithmetic structures in FHE, leading to inefficiencies.

Verifiable FHE. Recent research has advanced theoretical constructions for verifiable FHE, yet several challenges remain unaddressed. A line of work [17, 18, 19, 20, 21, 22, 23] supports only basic FHE operations such as LWE additions without bootstrapping, preventing fully homomorphism. [53, 54] show that verifiable FHE is possible with SNARKs. [55] provides a construction by encoding complete FHE using standard R1CS representations, but this method does not natively support the field arithmetic of FHE. To address this, [56] introduces R1CS defined over rings, which aligns with FHE arithmetic. However, these methods offer only theoretical results without practical constructions. Moreover, they rely on general constraint systems such as R1CS to represent FHE operations and leverage general-purpose SNARKs, which are not tailored for FHE. In contrast, our work directly proves the relations of FHE's core operations.

Recent work focuses on constructing concrete SNARK protocols for proving FHE. One approach is to use high-level languages, such as Rust, to implement FHE and then compile it into SNARK proofs using zkVM frameworks [24, 25]. However, this method suffers from extremely low efficiency, with proof generation taking several days. Alternatively, [26] applies Plonk [14] to directly prove FHE computations, reducing proving time to around half an hour. Yet, general-purpose SNARKs lack optimization for the specialized relations involved in FHE, limiting their efficiency in this context. In contrast, our work introduces a concrete SNARK tailored specifically for FHE, capable of generating proofs in seconds, achieving significant improvements in both speed and practical applicability.

Another line of work [57, 58, 59, 60] performs the integrity check on the plaintext instead of checking the computation result of ciphertexts. First, these approaches are not publicly verifiable so only the users with FHE secret keys can decrypt and verify the result. This is not desirable in decentralized applications [61, 62], where the result on the chain needs to be verified by anyone. Second, these approaches are vulnerable to active attacks [63] on FHE.

8. Conclusion

We propose HasteBoots, a tailored argument system for FHE, significantly reducing bootstrapping proof generation time. HasteBoots demonstrates scalable and efficient verifiable FHE, advancing privacy-preserving computation.

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM STOC*, M. Mitzenmacher, Ed. ACM Press, May / Jun. 2009, pp. 169–178.

[2] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, Cham, Dec. 2017, pp. 409–437.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[4] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Berlin, Heidelberg, Aug. 2012, pp. 868–886.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 309–325.

[6] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *EUROCRYPT 2015, Part I*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, Berlin, Heidelberg, Apr. 2015, pp. 617–640.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[8] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Berlin, Heidelberg, May 2016, pp. 305–326.

[9] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *ICALP 2018*, ser. LIPIcs, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Schloss Dagstuhl, Jul. 2018, pp. 14:1–14:17.

[10] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward, "Marlin: Preprocessing zkSNARKs with universal and updatable SRS," in *EUROCRYPT 2020, Part I*, ser. LNCS, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, Cham, May 2020, pp. 738–768.

[11] J. Bootle, A. Chiesa, Y. Hu, and M. Orru, "Gemini: Elastic snarks for diverse environments," in *EUROCRYPT*, 2022.

[12] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 733–764.

[13] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *CRYPTO*, 2020.

[14] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Report 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953

[15] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "HyperPlonk: Plonk with linear-time prover and high-degree custom gates," in *EUROCRYPT 2023, Part II*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14005. Springer, Cham, Apr. 2023, pp. 499–530.

[16] A. Golovnev, J. Lee, S. T. V. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic SNARKs for R1CS," in *CRYPTO 2023, Part II*, ser. LNCS, H. Handschuh and A. Lysyanskaya, Eds., vol. 14082. Springer, Cham, Aug. 2023, pp. 193–226.

[17] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010.

[18] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "How to run turing machines on encrypted data," in *CRYPTO*, 2013.

[19] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *CCS*, 2014.

[20] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez, "Rinocchio: Snarks for ring arithmetic," *JoC*, 2023.

[21] D. Fiore, A. Nitulescu, and D. Pointcheval, "Boosting verifiable computation on encrypted data," in *PKC*, 2020.

[22] A. Bois, I. Cascudo, D. Fiore, and D. Kim, "Flexible and efficient verifiable computation on encrypted data," in *PKC*, 2021.

[23] A. Viand, C. Knabenhans, and A. Hithnawi, "Verifiable fully homomorphic encryption," *arXiv:2301.07041*, 2023.

[24] T. R. Z. Team, "Universal zero knowledge," https://risczero.com/.

[25] T. SuccinctLabs, "The fastest, most feature-complete zkvm for developers," https://github.com/succinctlabs/sp1.

[26] L. T. Thibault and M. Walter, "Towards verifiable fhe in practice: Proving correct execution of tfhe's bootstrapping using plonky2," *ePrint*, 2024.

[27] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *EUROCRYPT 2010*, ser. LNCS, H. Gilbert, Ed., vol. 6110. Springer, Berlin, Heidelberg, May / Jun. 2010, pp. 1–23.

[28] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[29] S. Setty, J. Thaler, and R. Wahby, "Unlocking the

lookup singularity with lasso," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 180–209.

[30] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[31] A. Satriawan, R. Mareta, and H. Lee, "A complete beginner guide to the number theoretic transform (ntt)," *Cryptology ePrint Archive*, 2024.

[32] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic SNARKs for R1CS," Cryptology ePrint Archive, Paper 2021/1043, 2021. [Online]. Available: https://eprint.iacr.org/2021/1043

[33] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Annual Cryptology Conference*. Springer, 2013, pp. 71–89.

[34] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *37th ACM STOC*, H. N. Gabow and R. Fagin, Eds. ACM Press, May 2005, pp. 84–93.

[35] D. Micciancio and Y. Polyakov, "Bootstrapping in FHEW-like cryptosystems," Cryptology ePrint Archive, Report 2020/086, 2020. [Online]. Available: https://eprint.iacr.org/2020/086

[36] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *Journal of the ACM (JACM)*, vol. 39, no. 4, pp. 859–868, 1992.

[37] U. Haböck, "Multivariate lookups based on logarithmic derivatives," *Cryptology ePrint Archive*, 2022.

[38] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *Algorithmica*, vol. 12, pp. 225–244, 1994.

[39] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[40] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in *EUROCRYPT 2020, Part I*, ser. LNCS, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, Cham, May 2020, pp. 769–793.

[41] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for SNARKS and proof-carrying data," in *45th ACM STOC*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM Press, Jun. 2013, pp. 111–120.

[42] Z. Liu, D. Micciancio, and Y. Polyakov, "Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping," in *ASIACRYPT 2022, Part II*, ser. LNCS, S. Agrawal and D. Lin, Eds., vol. 13792. Springer, Cham, Dec. 2022, pp. 130–160.

[43] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE," in *ASIACRYPT 2021, Part III*, ser. LNCS, M. Tibouchi and H. Wang, Eds., vol. 13092. Springer, Cham, Dec. 2021, pp. 670–699.

[44] C. Bonte, I. Iliashenko, J. Park, H. V. L. Pereira, and N. P. Smart, "FINAL: Faster FHE instantiated with NTRU and LWE," in *ASIACRYPT 2022, Part II*, ser. LNCS, S. Agrawal and D. Lin, Eds., vol. 13792. Springer, Cham, Dec. 2022, pp. 188–215.

[45] Y. Lee, D. Micciancio, A. Kim, R. Choi, M. Deryabin, J. Eom, and D. Yoo, "Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption," in *EUROCRYPT 2023, Part III*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14006. Springer, Cham, Apr. 2023, pp. 227–256.

[46] N. Gama, M. Izabachène, P. Q. Nguyen, and X. Xie, "Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Berlin, Heidelberg, May 2016, pp. 528–558.

[47] Z. Zhang, T. Plantard, and W. Susilo, "Reaction attack on outsourced computing with fully homomorphic encryption schemes," in *ICISC 11*, ser. LNCS, H. Kim, Ed., vol. 7259. Springer, Berlin, Heidelberg, Nov. / Dec. 2012, pp. 419–436.

[48] Z. Li, S. D. Galbraith, and C. Ma, "Preventing adaptive key recovery attacks on the GSW levelled homomorphic encryption scheme," in *ProvSec 2016*, ser. LNCS, L. Chen and J. Han, Eds., vol. 10005. Springer, Cham, Nov. 2016, pp. 373–383.

[49] B. Chaturvedi, A. Chakraborty, A. Chatterjee, and D. Mukhopadhyay, "vr^{2}FHE- securing FHE from reaction-based key recovery attacks," Cryptology ePrint Archive, Report 2023/561, 2023. [Online]. Available: https://eprint.iacr.org/2023/561

[50] ——, ""Ask and thou shall receive": Reaction-based full key recovery attacks on FHE," in *ESORICS 2024, Part IV*, ser. LNCS, J. Garcia-Alfaro, R. Kozik, M. Choraś, and S. Katsikas, Eds., vol. 14985. Springer, Cham, Sep. 2024, pp. 457–477.

[51] I. Chillotti, N. Gama, and L. Goubin, "Attacking FHE-based applications by software fault injections," Cryptology ePrint Archive, Report 2016/1164, 2016. [Online]. Available: https://eprint.iacr.org/2016/1164

[52] A. Viand, C. Knabenhans, and A. Hithnawi, "Verifiable fully homomorphic encryption," 2023. [Online]. Available: https://arxiv.org/abs/2301.07041

[53] M. Manulis and J. Nguyen, "Fully homomorphic encryption beyond ind-cca1 security: Integrity through verifiability," in *EUROCRYPT*, 2024.

[54] S. Canard, C. Fontaine, D. H. Phan, D. Pointcheval, M. Renard, and R. Sirdey, "Relations among new cca security notions for approximate fhe," *ePrint*, 2024.

[55] S. Atapoor, K. Baghery, H. V. Pereira, and J. Spiessens, "Verifiable fhe via lattice-based snarks," *ePrint*, 2024.

[56] M.-Y. M. Huang, B. Li, X. Mao, and J. Zhang, "Fully homomorphic encryption with efficient public verification," *ePrint*, 2024.

[57] S. Garg, A. Goel, and M. Wang, "How to prove

statements obliviously?" in *CRYPTO*, 2024.
[58] D. F. Aranha, A. Costache, A. Guimarães, and E. Soria-Vazquez, "Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical," *ePrint*, 2023.
[59] S. Chatel, C. Knabenhans, A. Pyrgelis, C. Troncoso, and J.-P. Hubaux, "Verifiable encodings for secure homomorphic analytics," *arXiv:2207.14071*, 2022.
[60] ——, "Poster: Verifiable encodings for maliciously-secure homomorphic encryption evaluation," in *CCS*, 2023.
[61] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *FC*, 2020.
[62] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: Toward a secure {Data-Sharing} system from decentralized trust," in *NSDI*, 2020.
[63] I. Chillotti, N. Gama, and L. Goubin, "Attacking fhe-based applications by software fault injections," *ePrint*, 2016.
[64] S. Agrawal and D. Lin, Eds., *ASIACRYPT 2022, Part II*, ser. LNCS, vol. 13792. Springer, Cham, Dec. 2022.
[65] A. Canteaut and Y. Ishai, Eds., *EUROCRYPT 2020, Part I*, ser. LNCS, vol. 12105. Springer, Cham, May 2020.
[66] M. Fischlin and J.-S. Coron, Eds., *EUROCRYPT 2016, Part II*, ser. LNCS, vol. 9666. Springer, Berlin, Heidelberg, May 2016.

Appendix A. Additioanl Details on Modulus Switching

A.1. Modulus Switching with Floor

Given an input $a \in \mathbb{F}$, modulus switching with the floor is defined as follows

$$b = \left\lfloor \frac{a \cdot q}{Q} \right\rfloor$$

where the floor operation returns the greatest integer less than or equal to the fractional number $a \cdot q/Q$. Assuming $q|Q-1$, which is guaranteed in the FHEW/TFHE schemes, and defining $k = \frac{Q-1}{q}$, the modulus switching can be considered as a structured mapping. Specifically, it maps $a \in [0, k]$ to 0, maps $a \in [b \cdot k + 1, (b+1) \cdot k]$ for each b to $b \in \mathbb{Z}_q$.

Lemma 2. *Let q be a power of two and Q a prime such that $q|Q-1$. Define $k = \frac{Q-1}{q}$ such that $Q = 2kq + 1$. For any $a \in \mathbb{F}_Q$ and $b \in \mathbb{Z}_q$, we have $b = \left\lfloor \frac{a \cdot q}{Q} \right\rfloor$ if and only if either $a = 0$ and $b = 0$ or $a \in [b \cdot k + 1, (b+1) \cdot k]$.*