# HasteBoots: Proving FHE Bootstrapping in Seconds

Fengrun Liu[*,1], Haofei Liang[2], Tianyu Zhang[2], Yuncong Hu[†,2], Xiang Xie[3], Haisheng Tan[4], Yu Yu[2]

[1]*Shanghai Qi Zhi Institute,* [2]*Shanghai Jiao Tong University,* [3]*Shanghai Boduo Technology Co., Ltd,* [4]*University of Science and Technology of China*

## 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 NAND with bootstrapping, significantly outperforming existing methods. Our approach demonstrates the potential for scalable and efficient verifiable FHE, paving the way for practical, privacy-preserving computations.

## KEYWORDS

Succinct Arguments, Verifiable FHE

## 1 INTRODUCTION

Fully Homomorphic Encryption (FHE) [9, 10, 19, 21, 28, 29, 38] 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 [38]. 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.

This is where Succinct Non-Interactive Arguments of Knowledge (SNARKs) [4, 7, 18, 20, 33, 41, 42, 56, 66] 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 [5, 30, 31, 35, 37, 39, 62] managed to prove basic FHE operations, such as addition and multiplication, these methods fail to support complex FHE operations, particularly bootstrapping; thus, these works cannot prove general FHE circuits. 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) [58, 59], 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 bootstrapping. Consequently, existing zkVMs like RISC0 [59] and SP1 [58] require approximately **three days** and **one day**, respectively, to generate a proof for a single bootstrapping.

A recent work [61] uses general-purpose SNARKs to directly verify FHE operations. [61] 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 Number Theoretic Transform (NTT), accumulator update, modulus switching, and conversion 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 SNARKs, leading to efficiency loss. As a result, [61] needs about **half an hour** for proving the bootstrapping procedure, which limits the practical scalability of verifiable FHE.

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

∗ This work was completed during the author's internship at Shanghai Qi Zhi Institute.
† Corresponding author

## 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**. Our proof system targets FHE NAND with bootstrapping, as it is general enough to capture any FHE circuit. NAND is universal, and bootstrapping enables unbounded-depth homomorphic computations. We identify key atomic operations in the FHE NAND workflow as shown in Fig. 1 and build a new framework for proof construction as shown in Fig. 2. Based on this framework, we analyze FHE algorithms and identify FHEW/TFHE [21, 28] 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 generation. Our framework provides a foundation for future work to optimize proofs for FHE where it suffices to improve the subprotocols for each component of the framework.
- **We construct an efficient succinct argument HasteBoots for FHEW/TFHE**. We develop custom PIOPs optimized specifically for all internal FHE operations, leveraging their algebraic structure to significantly improve proof efficiency. This completeness enables an efficient PIOP for verifying the correctness of a functionally complete gate–FHE NAND with bootstrapping–which suffices to support arbitrary FHE computation. In particular, we achieve an optimal prover for the fast-NTT algorithm with bit-reversal, a widely used technique that may be of independent interest. We further propose an optimized PIOP for batched NTT operations on monomials, where the subclaim reduces to a single query on a sparse coefficient matrix, eliminating the need to prove the lift step typically required in FHE. Additionally, our modulus switching PIOP is novel, reducing the argument to simpler range checks without introducing additional noise. To balance proving efficiency with field-agnostic support across different settings, we select a PCS based on linear codes [41] and apply a packing mechanism to accelerate the opening on numerous small polynomials. 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 an FHE NAND operation with bootstrapping in **5 seconds**, whereas the state-of-the-art [61] requires 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. FHE NAND is a universal gate, including four main steps: LWE addition, batched NTT operations on monomials, accumulator update, and modulus switching.

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

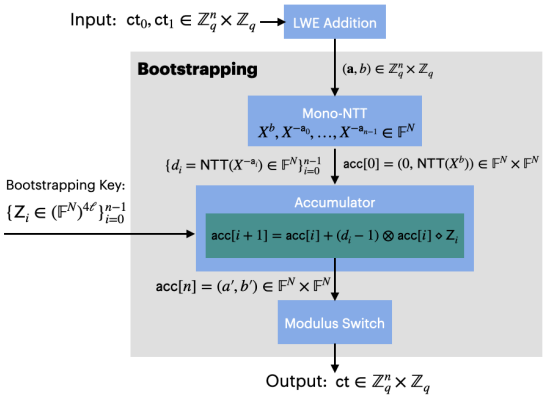


**Figure 1: The computation flow of the NAND operation in FHE. The gray part represents the bootstrapping procedure, where the bootstrapping keys are public. We assume $q = 2N$ for simplicity. $\otimes$ and $\diamond$ are defined in Section 2.2.**

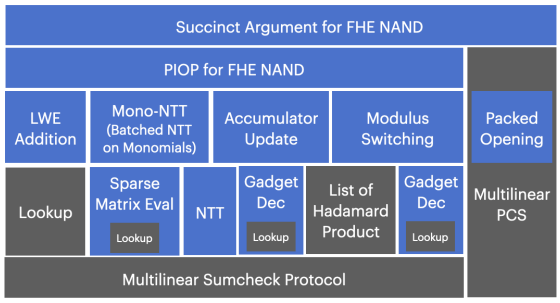


**Figure 2: Protocol design overview of proof systems for FHE NAND, highlighting our newly designed protocols in blue and existing protocols in gray.**

The FHE scheme first performs LWE addition on these ciphertexts to produce $(\boldsymbol{a}, b) = (\boldsymbol{a}_0 + \boldsymbol{a}_1, b_0 + b_1)$, with increased noise. Here, the term $\frac{5q}{8}$ is omitted for simplicity. Bootstrapping aims to reduce this noise by homomorphically decrypting the ciphertext using a circular encryption of the secret key—i.e., the secret key encrypted under itself. This is denoted by $\mathsf{Z}_i$, an $\mathbb{RGSW}$ ciphertext that encrypts the secret $s_i$. The decryption circuit for the LWE ciphertext $(\boldsymbol{a}, b)$ evaluates $\lfloor \frac{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle}{q/4} \rceil$.

Our bootstrapping procedure uses an RLWE scheme [51] over a cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, with its quotient ring $\mathcal{R}_Q = \mathcal{R}/(QR) = \mathbb{F}[X]/(X^N + 1)$, where $\mathbb{F}$ is the finite field over the prime $Q$. The cyclotomic ring contains a cyclic subgroup $G = \{X, \ldots, X^{2N}\}$, and a $2N$-th root of unity $\omega$ such that $\omega^{2N} = 1$. To perform the homomorphic decryption in the RLWE scheme, we lift the decryption circuit into the exponent, computing $X^{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle}$. Assuming $q = 2N$ for simplicity, the lift step 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 in $\mathbb{Z}_q$.

**PIOP Design**. We introduce our PIOP design as shown in Fig. 2 tailored for the FHE NAND workflow. We use 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 a different modulus $Q$, we need to emulate this modulus operation within the field. Given the multilinear polynomials $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, the relation $\boldsymbol{c}(x) = \boldsymbol{a}(x) + \boldsymbol{b}(x) \mod q$ holds for all $x \in \{0,1\}^{\log n}$ if and only if there exists $\boldsymbol{k}$ such that $\boldsymbol{a}(x) + \boldsymbol{b}(x) = \boldsymbol{k}(x) \cdot q + \boldsymbol{c}(x)$ and $\boldsymbol{k}(x) \in \{0,1\}$. This can be done efficiently using the multilinear sum-check protocol. We also need to prove that $\boldsymbol{c}(x)$ 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 $(0, \ldots, q-1)$.

**Mono-NTT: Batched NTT on Monomials**. We propose an efficient PIOP for batched NTT operations on monomials with an optimal prover. This protocol enables a direct reduction of the argument to a query on the output of the LWE addition, thereby eliminating the need to explicitly prove the *lifting* step—i.e., converting a ring element $b \in \mathbb{Z}_q$ into the exponent of a monomial $X^b$ in the cyclic subgroup $G$ (with exponent $b \cdot \frac{2N}{q}$, for simplicity, we assume $q = 2N$), as illustrated in Fig. 1. In FHE, the polynomial ring $\mathcal{R}_Q$ is defined over the irreducible polynomial $X^N + 1$, meaning that the lifted polynomial becomes $X^b$ if $b < N$, and $-X^{b-N}$ otherwise. Establishing a direct relation between a ring element and its corresponding lifted polynomial is nontrivial, making it challenging to prove with existing techniques.

Crucially, we observe that the lifted polynomial is always a monomial—i.e., a polynomial with exactly one non-zero coefficient–whose exponent is determined by the input of the lift operation. Moreover, these monomials are only multiplied with other polynomials during the subsequent accumulator updates. Thus, instead of explicitly proving the lift operation, we prove batched NTT operations on $M$ such monomials. Each monomial can be compactly represented by its coefficient and exponent: let $\boldsymbol{k} \in \mathbb{F}^M$ store the non-zero coefficients and $\boldsymbol{r} \in \mathbb{Z}_N^M$ store the corresponding exponents. This structure allows us to design an efficient batched NTT argument with optimal prover complexity $O(N + M)$, significantly improving over the naive $O(MN)$ approach.

Concretely, we randomize $M$ NTT relations into a single NTT relation by leveraging the linearity of the NTT. Recall that the NTT maps a polynomial's coefficient vector to its evaluations at $N$ roots of unity. Given $M$ such NTT relations (one per monomial), we construct a single randomized NTT relation via a random linear combination, using randomness $r' \in \mathbb{EF}^{\log M}$ (see Section 4.2 for details). This randomized relation can be computed in linear time, as each monomial has only one non-zero coefficient. It can also be verified in linear time via an NTT argument (further discussed in the next part), which reduces to querying the coefficient vector at a random point $r \in \mathbb{EF}^{\log M}$. Notably, this query corresponds to an evaluation at the joint point $(r, r')$ on a sparse $N \times M$ coefficient matrix, where each column is the full coefficient vector of one monomial. Prior works [20, 56, 57] observe that sparse matrices can be efficient represented using three vectors—row, col and val—to locate the non-zero entries. In our case, this representation simplifies further: each column has exactly one non-zero value $k_i$ at position $r_i$. Therefore, the evaluation proof at $(r, r')$ reduces to queries on row $= \boldsymbol{r}$ and val $= \boldsymbol{k}$, both directly derived from the output of the LWE addition. The column vector col $= (0, \ldots, M)$ is fixed and can be omitted.

**Accumulator Update**. The accumulator is initialized with an $\mathbb{RLWE}$ ciphertext acc$[0]$. Each update takes a circular encryption $\mathsf{Z}_i$ 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 Hadamard products, NTT operations, and gadget decompositions with respect to a tunable basis $B$. Hadamard products can be efficiently proven using sum-check protocols.

The gadget decomposition generalizes bit decomposition by breaking down the coefficients $\boldsymbol{f} \in \mathbb{F}^N$ of a polynomial into $\ell = \lceil \log_B Q \rceil$ 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 in FHE adopts a larger basis $B$ to improve computational efficiency. However, this comes at the cost of proof efficiency, as the proving time grows quadratically with the degree. On the upside, a larger basis reduces the number of NTT operations and Hadamard products over the decomposed gadget polynomials, which can lead to a more efficient prover overall. To balance computational and proof efficiency, we choose a larger basis $B$ and employ a lookup protocol to verify that $\boldsymbol{f}_i(x)$ values fall within the range $[0, B)$.

The NTT operation, defined over $\mathcal{R}_Q$ with a $2N$-th root of unity $\omega$, transforms a degree-$N$ polynomial's coefficient vector into an evaluation vector at $N$ specially chosen points. This transformation can be efficiently computed in $O(N \log N)$ using the butterfly algorithm, known as fast-NTT [55]. More specifically, the NTT can be expressed using a Fourier-like matrix $F(\mathcal{Y}, \mathcal{X}) = \omega^{(2\mathcal{Y}+1)\cdot\mathcal{X}}$. Prior work [48] presents a linear-time argument for the FFT, based on the standard Fourier matrix defined by roots of unity. However, these techniques do not directly apply to the NTT, as its use of different roots of unity changes the structure of the Fourier matrix. Moreover, the fast-NTT algorithm employs a divide-and-conquer butterfly structure that outputs the evaluation vector in bit-reversed order—an aspect also exhibited in the FFT but overlooked in [48]. Consequently, their prover incurs an additional $O(N \log N)$ overhead to reorder the evaluation vector, degrading proof generation performance.

In this work, we propose a PIOP for the NTT operation that achieves optimal prover efficiency and, notably, explicitly accounts for bit-reversal. Specifically, we introduce a new Fourier-like matrix $F^{\mathsf{R}}(\mathcal{Y}, \mathcal{X}) = \omega^{(2\mathcal{Y}^{\mathsf{R}}+1)\cdot\mathcal{X}}$, which inherently encodes bit-reversal. Here, $\mathcal{Y}^{\mathsf{R}}$ denotes the bit-reversal of $\mathcal{Y}$ (see Section 2.2 for its formal definition). This provides an exact expression for the NTT

relation between a coefficient vector $\boldsymbol{c}$ and its bit-reversed output $\boldsymbol{a}^{\mathsf{R}}$, given by $\boldsymbol{a}^{\mathsf{R}} = F^{\mathsf{R}} \cdot \boldsymbol{c}$. Rewriting this relation in terms of multivariate polynomials, it suffices to verify the equality $\tilde{\boldsymbol{a}}^{\mathsf{R}}(u) = \sum_{x\in\{0,1\}^{\log N}} \tilde{\boldsymbol{c}}(x)\tilde{F}^{\mathsf{R}}(u,x)$ at a random point $u \in \mathbb{F}^{\log N}$ using the sum-check protocol. However, the prover requires quadratic time to compute the bookkeeping table $\tilde{F}^{\mathsf{R}}(u,x)$ for all $x \in \{0,1\}^{\log N}$ to generate the sum-check messages with techniques from [66].

To achieve an optimal prover, we decompose the exponents into bitwise components and break the computation of $\tilde{F}^{\mathsf{R}}(u,x)$ into logarithmic rounds. The key insight is that the term $\omega^{2^{\log N-i}} = \omega^{\frac{2N}{2^{i+1}}}$ corresponds to the $2^{i+1}$-th roots of unity, denoted by $\omega_{2^{i+1}}$, meaning that $\omega_{2^{i+1}}^{\mathcal{X}}$ takes only $2^{i+1}$ distinct values for all $\mathcal{X}$. Hence, we decompose $\mathcal{X}$ and $\mathcal{Y}^{\mathsf{R}}$ so that in the $i$-th round, the prover computes only $O(2^i)$ values. While [48] employs a similar idea, our decomposition method differs in structure (see Section 3.2 for details). This approach enables the prover to use a dynamic programming algorithm to compute the bookkeeping table in linear time, yielding a strictly linear-time prover that inherently supports bit-reversal.

Moreover, leveraging the linearity of the NTT operation, we propose a batching technique: instead of proving each NTT relation individually, we compute a single randomized NTT relation as a random linear combination of the $M$ original NTT relations, and verify this combined relation using our NTT argument. As described in Mono-NTT, this technique yields an asymptotic improvement for the prover in the case of monomials, as $M$ monomials contain only $M$ non-zero coefficients. In addition, even when verifying NTT relation for general polynomials, this approach offers concrete performance gains in protocols over extension fields: it reduces $O(MN)$ extension field operations to $O(MN)$ multiplications between the base field and the extension field.

**Modulus Switching**. The final step in bootstrapping is modulus switching, which converts an $\mathbb{LWE}$ ciphertext under modulus $Q$ to a new $\mathbb{LWE}$ ciphertext under a smaller modulus $q$. Specifically, this involves rounding $\frac{a\cdot q}{Q}$ to the nearest integer in $\mathbb{Z}_q$, as follows: $b = \lfloor \frac{a\cdot q}{Q} \rceil \mod q$. The intricate arithmetic of this operation makes proof generation challenging. A straightforward approach using lookup arguments to implement it as a custom gate results in an insufficient prover due to the large table size of $Q$. Instead, we exploit the inherent relationship between $q$ and $Q$ in FHE parameters to transform this complex operation into a highly structured mapping: one that assigns a range of values to the same integer in $\mathbb{Z}_q$ with the range itself parameterized by that output value. This mapping can then be verified using simpler range-check arguments.

Specifically, assuming that $2q \mid Q-1$, we define $k = (Q-1)/2q$, giving the relation $Q = 2kq+1$. This structure allows us to partition the field into $q$ ranges of length $2k$, along 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 $b \in \{1,\ldots,q-1\}$. The remaining uncovered range, $a \in [0,k]\cup(0,-k] \equiv Q$ is mapped to 0. Although this range consists of two separate ranges, they can be viewed as 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 ensure a consistent mapping, we introduce an auxiliary witness $b' \in \{1,\ldots,q\}$ such that $b' \equiv b \mod q$ to parameterize the range. As a result, this mapping can be expressed as a dichotomy: either $a = k$ and $b = 0$, or for any $a \in [(2b'-1)\cdot k+1, (2b'+1)\cdot k]$, we have $a$ mapped to $b$ where $b \in \mathbb{Z}_q$, and $b' \in \{1,\ldots,q\}$ satisfies $b' \equiv b \mod q$.

The structured mapping relation extracted from the modulus switching can be efficiently proven by standard techniques. Moreover, it faithfully emulates the modulus switching without introducing additional error into FHE ciphertexts, making it more flexible to different $Q$. In contrast, [61] introduces extra noise to FHE ciphertexts, limiting its applicability when $Q$ is small.

**Polynomial Commitments**. Finally, to commit multilinear polynomials, we employ polynomial commitment schemes (PCS) compatible with FHE operations. We select Brakedown [40] 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 rely on the relative code distance. Furthermore, Brakedown's opening proofs are not homomorphic, 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 evaluated at the corresponding random sub-point, reducing the overall cost.

## 2 PRELIMINARY

### 2.1 Notation

We denote the set $\{0,\ldots,a-1\}$ by $[a]$. Let $\mathbb{Z}_q$ be a ring over an integer $q$, and $\mathbb{F}_Q$ a field defined over the prime $Q$ (omitted where clear). For a vector $\boldsymbol{a} \in \mathbb{F}^N$, the $i$-th entry is denoted by $a_i$. We use $\boldsymbol{a}[n_1..n_2]$ to denote the sub-vector of $\boldsymbol{a}$ starting at index $n_1$ and ending before index $n_2$ (i.e., excluding $n_2$). The vector $\boldsymbol{a}$ can be decomposed with respect to a basis $B \in \mathbb{Z}$ into $\ell = \lceil \log_B Q \rceil$ vectors $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1}$ such that $\boldsymbol{a} = \sum_{i=0}^{\ell-1} B^i \cdot \boldsymbol{a}_i$ where $\boldsymbol{a}_0,\ldots,\boldsymbol{a}_{\ell-1} \in \mathbb{Z}_B^N$. This decomposition is referred to as a gadget decomposition and is denoted by $\mathrm{Dec}(\boldsymbol{a}; B, \ell) = (\boldsymbol{a}_0,\ldots,\boldsymbol{a}_\ell)$. 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 $X \in \mathbb{F}$ denotes a field variable. For $x = (x_0,\ldots,x_{\log N-1})$, define the canonical injection to-fld$(x) = \sum_{i=0}^{\log N-1} 2^i \cdot x_i$, which maps $x \in \{0,1\}^{\log N}$ to $\mathbb{F}$, and let to-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 $\boldsymbol{a}^{\mathsf{R}} = (a_{0^{\mathsf{R}}},\ldots,a_{(N-1)^{\mathsf{R}}}) \in \mathbb{F}^N$.

**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 $\boldsymbol{a}(x) = a_X$. 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 the $2N$-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 represented by the coefficient vector $\boldsymbol{c} = (c_0, \ldots, c_{N-1}) \in \mathbb{F}^N$.

**NTT/INTT**. The Number Theoretic Transform (NTT) enables polynomial multiplication in $\mathcal{R}_Q$ via the fast-NTT algorithm [55]. 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 at $X = \omega^{2\mathcal{Y}+1}$ for $\mathcal{Y} \in [N]$. Since the fast-NTT produces $\boldsymbol{a}$ in bit-reversed order, we define $\mathsf{NTT}(\boldsymbol{c}) = \boldsymbol{a}^{\mathrm{R}}$, and its inverse as $\mathsf{INTT}(\boldsymbol{a}^{\mathrm{R}}) = \boldsymbol{c}$.

**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

We focus on the FHEW-like fully homomorphic encryption systems with fast bootstrapping procedures. A Learning with Errors (LWE) [54] 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 sampled from some discrete gaussian distribution with small norm, and $m \in \mathbb{Z}_t$ is the message. To define the relation among various FHE 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 $\mathrm{ct} = (\boldsymbol{a}, b) \in \mathbb{LWE}$.

A Ring Learning with Errors (RLWE) [51] 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 a 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 $\mathrm{c} \in \mathbb{RLWE}$, containing two vectors $\mathrm{c}.\boldsymbol{a} \in \mathbb{F}^N$ and $\mathrm{c}.\boldsymbol{b} \in \mathbb{F}^N$.

Let $B$ be the basis for decomposition and $\ell = \lceil \log_B Q \rceil$. We define an RLWE′ ciphertext as containing $\ell$ RLWE ciphertexts that encrypt the messages $\boldsymbol{m}, B\boldsymbol{m}, ..., B^{\ell-1}\boldsymbol{m}$ under the same secret key. We define an RGSW ciphertext as containing 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 $\mathrm{Z} \in \mathbb{RGSW}$ as containing $4\ell$ vectors $\mathrm{Z}.\boldsymbol{a}_0, \ldots, \mathrm{Z}.\boldsymbol{a}_{2\ell-1}$ and $\mathrm{Z}.\boldsymbol{b}_0, \ldots, \mathrm{Z}.\boldsymbol{b}_{2\ell-1}$. Let $\mathrm{Z}.\vec{\boldsymbol{a}} = (\mathrm{Z}.\boldsymbol{a}_0, \ldots, \mathrm{Z}.\boldsymbol{a}_{2\ell-1})$ and $\mathrm{Z}.\vec{\boldsymbol{b}} = (\mathrm{Z}.\boldsymbol{b}_0, \ldots, \mathrm{Z}.\boldsymbol{b}_{2\ell-1})$.

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

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

This operation is often referred to as the external product.

We provide a brief overview of the entire FHE NAND with bootstrapping in Algorithm 3 and Appendix A and refer the reader to [53] for detailed explanations.

## 2.3 Useful PIOP

**Sum-check Protocol**. Let $\mathcal{G}$ be a $\mu$-variate polynomial defined over a field $\mathbb{F}$. The goal of the sum-check protocol is to allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of a sum: $T = \sum_{x\in\{0,1\}^\mu} \mathcal{G}(x_1, x_2, \ldots, x_\mu)$. Lund et al. [50] proposed a sum-check protocol that allows the verifier to efficiently verify the claimed sum. The protocol results in linear proving time and logarithmic proof size by delegating the computation to the prover. See Appendix B.1 for the formal definition of the sum-check protocol.

**Hadamard Protocol**. We define a Hadamard protocol as a PIOP for the relation $\mathcal{R}_\circ$ (see Definition 1), which involves a sum of $M$ Hadamard products. The protocol construction (see Appendix B.2) follows directly from the sum-check protocol, achieving perfect completeness and soundness error $O(\log N)/|\mathbb{EF}|$. The prover runs in $O(MN)$ time, and the verifier runs in $O(\log N + M)$ where $M$ corresponds to the query complexity for evaluating $\mathcal{G}$. The communication cost of the protocol is $O(\log N)$ when the degree of $\mathcal{G}$ is constant, which holds in our constructions.

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

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

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

**Lookups Based on the Logarithmic Derivative**. Lookup arguments are used to prove that a sequence of values belongs to a prescribed table. Haböck proposed a lookup argument based on the logarithmic derivative in [43]. According to Lemma 5 in [43], there exists a PIOP protocol (described in Appendix B.3) for the batched lookup relation $\mathcal{R}_{\mathsf{lookup}}$ (see Definition 2) that achieves perfect completeness and soundness error $O(MN)/|\mathbb{EF}|$, with communication cost of $O(\log N + M)$ field elements. The prover runs in $O(MN)$, and the verifier in $O(\log N + M)$. For convenience, we also define a variant—the indexed lookup relation $\mathcal{R}_{\mathsf{id-lookup}}$ —which extends $\mathcal{R}_{\mathsf{lookup}}$ by attaching an index to each lookup query.

Def 2. *The batched lookup 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 $\boldsymbol{t}, \boldsymbol{f}_0, \ldots, \boldsymbol{f}_{M-1} \in \mathbb{F}^N$ and $\bigcup_{i=0}^{M-1}\{\boldsymbol{f}_i(x) \mid x \in \{0,1\}^{\log N}\} \subseteq \{\boldsymbol{t}(x) \mid x \in \{0,1\}^{\log N}\}$ (as sets).*

Def 3. *The indexed lookup relation* $\mathcal{R}_{\mathsf{id-lookup}}$ *consists of tuples*

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

*where* $\boldsymbol{t}, \boldsymbol{f} \in \mathbb{F}^N$, $\mathsf{id} \in \mathbb{Z}_N^N$ *and* $\{(\mathsf{id}(x), \boldsymbol{f}(x)) \mid x \in \{0,1\}^{\log N}\} \subseteq \{(\text{to-fld}(x), \boldsymbol{t}(x)) \mid x \in \{0,1\}^{\log N}\}$ *as sets.*

# 3 PIOP FOR BUILDING BLOCKS

## 3.1 Simplified Sparse Matrix Evaluation

[56, 57] proposed a standard commitment scheme for sparse multilinear polynomials with optimal prover costs, in which the sparse matrices are represented by dense polynomials. The evaluation proof in this case is essentially a specialized PIOP with dense polynomials being committed. In this section, we consider a specialized sparse $N \times M$ matrix $C$ in which each column contains exactly one non-zero entry. Building on Claim 1 from [56], we formalize the evaluation of this special matrix structure in Lemma 3.1 and define the corresponding relation, $\mathcal{R}_{\mathsf{sparse}}$, in Definition 4. Our construction, presented in Appendix C.1, simplifies the original approach by replacing the memory-checking technique with an indexed lookup argument, since the table size is small in our setting.

Lemma 3.1. *Given a* $\log N + \log M$*-variate multilinear polynomial* $\tilde{C}$ *defined over a sparse* $N \times M$ *matrix* $C$*, in which each column has only one non-zero entry, there exist two* $(\log M)$*-variate multilinear polynomials* row *and* col *such that the following holds for all* $r_x \in \mathbb{EF}^{\log N}, r_y \in \mathbb{EF}^{\log M}$*:*

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

Proof. The proof follows Claim 1 in [57], with $\mathsf{col}(k) = \text{to-fld}(k)$ for all $k \in \{0,1\}^{\log M}$. □

Def 4. *The sparse matrix 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* $r_x \in \mathbb{EF}^{\log N}$, $r_y \in \mathbb{EF}^{\log M}$, $e \in \mathbb{EF}$, $\mathsf{val} \in \mathbb{F}^M$, $\mathsf{row} \in \mathbb{Z}_M^M$, *and* $\sum_{k \in \{0,1\}^{\log M}} \mathsf{val}(k) \cdot \tilde{eq}\left(\text{to-bits}(\mathsf{row}(k)), r_x\right) \cdot \tilde{eq}\left(k, r_y\right) = e$.

Theorem 3.2. *For every finite field* $\mathbb{F}$ *and its extension field* $\mathbb{EF}$*, and for all positive integers* $N, M$*, there exists a PIOP for the relation* $\mathcal{R}_{\mathsf{sparse}}$ *that achieves perfect completeness and soundness error* $O(M)/|\mathbb{EF}|$*, with communication cost of* $O(\log M)$*. The prover runs in* $O(M)$ *time, while the verifier runs in* $O(\log M)$ *time.*

We prove it with Construction 3 in Appendix C.1

## 3.2 NTT/INTT

In this section, we propose a specialized PIOP for the NTT/INTT operation that inherently supports bit-reversal of the evaluation vector, achieving a strictly linear-time prover for NTT without additional overhead to reorder the outputs. Additionally, we propose an optimization for proving batched NTT operations by leveraging the linearity of the NTT operation.

In our work, the NTT operation transforms a polynomial's coefficient vector $\boldsymbol{c} \in \mathbb{F}^N$ into an evaluation vector $\boldsymbol{a} \in \mathbb{F}^N$, evaluated at $X = \omega^{2\mathcal{Y}+1}$ for $\mathcal{Y} \in [N]$, with the output vector arranged in a bit-reversed order, i.e., $\boldsymbol{a}^{\mathrm{R}}$. Specifically, the $\mathcal{Y}$-th entry of $\boldsymbol{a}^{\mathrm{R}}$ corresponds to the evaluation at $X = \omega^{2\mathcal{Y}^{\mathrm{R}}+1}$. Hence, the NTT operation can be expressed as a matrix-vector multiplication $\boldsymbol{a}^{\mathrm{R}} = F^{\mathrm{R}} \cdot \boldsymbol{c}$, where $F^{\mathrm{R}}$ is a Fourier-like matrix defined by $F^{\mathrm{R}}(\mathcal{Y}, \mathcal{X}) = \omega^{(2\mathcal{Y}^{\mathrm{R}}+1)\cdot\mathcal{X}}$. To prove this matrix-vector multiplication, we first rewrite the polynomial evaluation equation as a multivariate polynomial expression:

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

for $y \in \{0,1\}^{\log N}$. Given the evaluation $\tilde{\boldsymbol{a}}^{\mathrm{R}}(u)$ at a random point $u \in \mathbb{EF}^{\log N}$, the proof reduces to a sum-check protocol applied to Eq. (1). Thaler [60] introduces a dynamic programming algorithm that initializes the values of $\tilde{\boldsymbol{c}}(\mathrm{x})$ for all $x \in \{0,1\}^{\log N}$ in linear time. This precomputed table, referred to as the bookkeeping table in [66], enables the prover to generate sum-check messages efficiently. Prior work [48] proposes a linear-time algorithm for computing the bookkeeping table for $\tilde{F}(u, x)$ when proving FFT, where the Fourier matrix is defined by $F(\mathcal{Y}, \mathcal{X}) = \omega^{\mathcal{Y}\mathcal{X}}$ for $\mathcal{Y} \in [M], \mathcal{X} \in [N]$. However, this approach cannot be directly applied to compute the bookkeeping table for $\tilde{F}^{\mathrm{R}}(u, x)$ in the case of fast NTT, due to the distinct structure of the matrix $F^{\mathrm{R}}$.

Instead, we adopt a similar strategy by decomposing the exponents of the root of unity $\omega$. 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 $\omega_{2^{i+1}}^{\mathcal{X}}$ takes on $2^{i+1}$ distinct values for all $\mathcal{X} \in [N]$. We express $\tilde{F}^{\mathrm{R}}(u, x)$ as follows:

$$\begin{aligned}
\tilde{F}^{\mathrm{R}}(u, x) &= \sum_{y \in \{0,1\}^{\log N}} \tilde{eq}\,(u, y)\, \tilde{F}^{\mathrm{R}}(y, x) && (2)\\
&= \sum_{y \in \{0,1\}^{\log N}} \tilde{eq}\,(u, y)\, \omega^{(2\mathcal{Y}^{\mathrm{R}}+1)\cdot\mathcal{X}} && (3)\\
&= \omega^{\mathcal{X}} \cdot \sum_{y \in \{0,1\}^{\log N}} \tilde{eq}\,(u, y)\, \omega^{\mathcal{X}\cdot 2\mathcal{Y}^{\mathrm{R}}} && (4)\\
&= \prod_{i=0}^{\log N - 1} (1 - u_i + u_i \cdot \omega_{2^{i+1}}^{\mathcal{X}}) \cdot \omega^{2^i \cdot x_i} && (5)
\end{aligned}$$

In Eq. (4), we decompose the bits of the first $\mathcal{X}$ and of $\mathcal{Y}^{\mathrm{R}}$, while keeping the second $\mathcal{X}$ intact. This enables us to express each evaluation into a grand product form. The full derivation is provided in Appendix E.2. Note that Eq. (5) includes an extra term, $\omega^{2^i \cdot x_i}$, in the grand product compared to Eq. (8) in [48], due to the different structure of the new Fourier-like matrix used in our NTT proof.

Using the property that $\omega_{2^{i+1}}^{\mathcal{X}}$ takes on $2^i + 1$ distinct values, we divide the computation of $\tilde{F}^{\mathrm{R}}(u, x)$ into $\log N$ rounds and compute the bookkeeping table 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 the $i$-th round, where $i$ ranges from 0 to $\log N - 1$.

**Reducing the verifier time**. In the final round of the sum-check protocol, the verifier must evaluate $\tilde{F}^{\mathrm{R}}(u, \cdot)$ at a random point $r \xleftarrow{\$} \mathbb{EF}^{\log N}$. Directly evaluating it via Algorithm 1 would require linear time for the verifier, so we delegate the computation to the prover using a sequence of sum-check protocols, reducing the verifier's runtime to $O(\log^2 N)$. Specifically, we adopt a delegation protocol inspired by [48], leveraging the fact that the computation

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

**Input:** The degree $N$ and the $2N$-th root of unity $\omega$.
**Input:** The random point $u \in \mathbb{EF}^{\log N}$.
**Output:** $A_{F^{\mathrm{R}}}$ storing $\tilde{F}^{\mathrm{R}}(u, x)$ for all $x \in \{0,1\}^{\log N}$

$A_{F^{\mathrm{R}}}[0] = 1$
**for** $i = 0, \ldots, \log N - 1$ **do**
  **for** $j = 2^{i+1} - 1, \ldots, 0$ **do**
    $b = (j >> i) \& 1$  // the MSB of $j$
    $A_{F^{\mathrm{R}}}[j] = A_{F^{\mathrm{R}}}[j \% 2^i] \cdot ((1 - u_i) + u_i \cdot \omega^j_{2^{i+1}}) \cdot \omega^{2^i \cdot b}$
**return** $A_{F^{\mathrm{R}}}$

in Algorithm 1 naturally forms a structured layered circuit. The delegation protocol consists of $\log N$ sum-check protocols, each corresponding to an iteration of the algorithm. In round $i$, verifying the correctness of array $A_{F^{\mathrm{R}}}$ of size $2^{i+1}$ reduces to checking an evaluation of the prior round's array of size $2^i$ at a random point. This recursive process terminates at the base case, where $A_{F^{\mathrm{R}}}[0] = 1$, as initialized in Step 1. See more details in Appendix E.1.

By delegating the computation of $\tilde{F}^{\mathrm{R}}(u, v)$, the verifier only needs to evaluate $\tilde{\boldsymbol{c}}(\cdot)$ at the random point $r$. Consequently, we view the full PIOP (including the delegation protocol) for the NTT relation $\mathcal{R}_{\mathsf{NTT}}$ as a mechanism that reduces the original check $\sum_{x \in \{0,1\}^{\log N}} \tilde{\boldsymbol{c}}(x) \cdot \tilde{F}^{\mathrm{R}}(u, x) \stackrel{?}{=} e$ to the simpler subclaim of $\tilde{\boldsymbol{c}}(r) \stackrel{?}{=} d$.

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

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

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

Theorem 3.3. *For every finite field $\mathbb{F}$ and its extension field $\mathbb{EF}$, and for all positive integers $N$, there exists a PIOP for relation $\mathcal{R}_{\mathsf{NTT}}$ that achieves perfect completeness and soundness error $O(\log^2 N)/|\mathbb{EF}|$, with communication cost of $O(\log^2 N)$. The prover runs in $O(N)$ time, and the verifier in $O(\log^2 N)$ time.*

Proof. Theorem 3.3 follows from a construction that combines a sum-check protocol—with a bookkeeping table computed by Algorithm 1—with a delegation protocol described in Appendix E.1. □

**Batched NTT Operations**. Leveraging the linearity of the NTT operation, we can efficiently prove multiple NTT instances by reducing them to a single randomized NTT check. This technique not only optimizes the protocol for batched NTT operations on monomials (see Section 4.2), but also yields concrete performance improvements in practice.

Suppose the prover wishes to convince the verifier of the correctness of $M$ NTT instances, each simply expressed as $(u, e_i, \boldsymbol{c}_i)$, where $\boldsymbol{c}_i$ is the $i$-th coefficient vector and $e_i = \sum_{x \in \{0,1\}^{\log N}} \boldsymbol{c}_i(x) \cdot \tilde{F}^{\mathrm{R}}(u, x)$. All instances share the same evaluation point $u \in \mathbb{EF}^{\log N}$. To batch these, we sample random scalars $\rho_0, \ldots, \rho_{M-1} \stackrel{\$}{\leftarrow} \mathbb{EF}$ and form a single randomized instance $(u, e', \boldsymbol{c}')$, where $\boldsymbol{c}' = \sum_{i=0}^{M-1} \rho_i \cdot \boldsymbol{c}_i$ and $e' = \sum_{i=0}^{M-1} \rho_i \cdot e_i$. Correctness follows from the linearity of NTT operation: $e' = \sum_{i=0}^{M-1} \rho_i \cdot \sum_x \boldsymbol{c}_i(x) \cdot \tilde{F}^{\mathrm{R}}(u, x) = \sum_x (\sum_{i=0}^{M-1} \rho_i \cdot \boldsymbol{c}_i(x)) \cdot \tilde{F}^{\mathrm{R}}(u, x) = \sum_x \boldsymbol{c}'(x) \cdot \tilde{F}^{\mathrm{R}}(u, x)$. The randomization introduces a soundness error of $1/|\mathbb{EF}|$, which is negligible. The prover computes the randomized coefficient vector and then proves a single NTT instance; the former step dominates the runtime for general (dense) coefficient vectors. However, if the coefficient vectors are sparse with only $O(M)$ non-zero entries, the prover time reduces to $O(M+N)$. This batching technique is especially beneficial when the protocol is executed over an extension field while the original NTT instances are defined over the base field. The prover can reduce $O(MN)$ extension field operations to $O(MN)$ base-to-extension field multiplications, significantly improving concrete performance.

Alternatively, instead of sampling $M$ random scalars, we may view $(e_0, \ldots, e_{M-1})$ as defining a multilinear extension and check the equality by evaluating it at a random point $r' \in \mathbb{EF}^{\log M}$. The same derivation holds with scalars $\rho_i = \tilde{eq}(\text{to-bits}(i), r')$. This variant incurs a soundness error of $\log M/|\mathbb{EF}|$ by Schwartz-Zippel lemma and requires only $\log M$ random elements. Used in our PIOP for batched NTT operations on monomials, this approach facilitates reducing the argument to evaluating a sparse $N \times M$ coefficient matrix at a joint random point $(r, r')$, where $r \in \mathbb{EF}^{\log N}$ is the query point derived from the subclaim of the NTT check on $(u, e', \boldsymbol{c}')$.

## 3.3 Gadget Decomposition

In this section, we present a PIOP for the operation $\mathrm{Dec}(\boldsymbol{a}; B, \ell)$ that decomposes a vector $\boldsymbol{a} \in \mathbb{F}^N$, with basis $B$, into $\ell$ 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 $\ell = \lceil \log_B Q \rceil$. Compared to bit decomposition, FHE typically adopts a relatively larger basis $B$ to improve computational efficiency. A larger basis also simplify the arithmetic circuit in the accumulator, reducing the number of NTT operations and Hadamard products over decomposed gadgets. To ensure the efficiency both in FHE computation and proof generation, we utilize the PIOP for the batched lookup relation $\mathcal{R}_{\mathsf{lookup}}$ (defined in Section 2.3) to check that the range of decomposed gadgets is within the table $\boldsymbol{t} = (0, \ldots, B-1)$.

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

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

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

Theorem 3.4. *For every finite field $\mathbb{F}$ and its extension field $\mathbb{EF}$, and for all positive integers $N, B$, there exists a PIOP for the relation $\mathcal{R}_{\mathsf{dec}}$ that achieves perfect completeness and soundness error $O(\ell N)/|\mathbb{EF}|$, with communication cost $\log N + \ell$. The prover runs $O(\ell N)$ time, and the verifier runs in $O(\log N + \ell)$ time.*

We prove it with Construction 4 in Appendix C.2.

**Extension to Range Check**. We generally use the lookup argument to prove $a$ lies in a range $[0, L)$, using 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 gadget decomposition to check the range relation $\mathcal{R}_{\mathsf{range}}$ (Definition 7), since the prover time for lookup arguments increases linearly with the table size. When the range size $L$ is not an exact power of $B$, let $\ell = \lceil \log_B L \rceil$. The prover computes the gadget decompositions of $\boldsymbol{a}$ and $\boldsymbol{a} + (B^\ell - L)$ and uses two invocations of the decomposition protocol to ensure that the intersection of the two verified ranges is exactly $[0, L)$. The detailed PIOP for $\mathcal{R}_{\mathsf{range}}$ is omitted here.

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

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

*where $\boldsymbol{a} \in \mathbb{Z}_L^N$ and $\ell = \lceil \log_B L \rceil$.*

## 4 PIOP FOR FHE OPERATIONS

In this section, we introduce our PIOPs for the operations involved in the FHE NAND circuit, as illustrated in Fig. 1.

### 4.1 LWE Addition

The core idea behind proving LWE addition is to emulate a straightforward addition modulo $q$ within our proof systems, which operates over modulus $Q$. Given $a, b \in \mathbb{Z}_q$, the result $c = a + b \bmod q$ 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 ensure that $c$ lies within the correct range, we employ a lookup argument using the table $\boldsymbol{t} = (0, \ldots, q-1)$. This is feasible in our design since the modulus $q$ used in the LWE scheme is relatively small.

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

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

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

Theorem 4.1. *For every finite field $\mathbb{F}$ and extension field $\mathbb{EF}$, and for all positive integers $q, N$, there exists a PIOP for the relation $\mathcal{R}_{\text{add}}$ with perfect completeness and soundness error $O(N)/|\mathbb{EF}|$. The prover runs in $O(N)$ time, and the verifier in $O(\log N)$ time.*

We prove it with Construction 5 in Appendix C.3.

### 4.2 Mono-NTT: Batched NTT on Monomials

In this section, we present a PIOP for batched NTT operations on monomials. This construction offers two key advantages. First, since each resulting polynomial after the lift step is a monomial with a single nonzero coefficient and is only multiplied with other polynomials during accumulator updates, we can leverage the sparsity of their coefficient vectors to efficiently prove batched NTT, as described in Section 3.2. Second, the argument for batched NTT on monomials reduces to querying the monomials' coefficients and exponents, which are directly derived from the output of the LWE addition. This allows us to eliminate the need to explicitly prove the lift step—i.e., converting a ring element into the exponent of a polynomial as illustrated in Fig. 1.

As previously shown in Fig. 1, the output of the LWE addition lies in the ring $\mathbb{Z}_q$. To homomorphically evaluate the decryption circuit in the RLWE scheme, we first lift each $a \in \mathbb{Z}_q$ into the exponent of a monomial $X^b \in G$, where $G = \{X, \ldots, X^{2N}\}$ is a cyclic subgroup of the polynomial ring $\mathcal{R}_Q = \mathbb{F}/(X^N + 1)$. Since $q \mid 2N$, this is done by computing $b = a \cdot \frac{2N}{q} \in \mathbb{Z}_{2N}$. To ensure correct representation within $\mathcal{R}_Q$, we reduce $X^b$ modulo $X^N + 1$ when $b \geq N$. The result of this modular reduction is $X^b$ if $b < N$ and $-X^{b-N}$ otherwise.

Although the resulting polynomials are monomials with coefficients of 1 or $-1$, we consider a general set of $M$ monomials of the form $\{k_0 \cdot X^{r_0}, k_1 \cdot X^{r_1}, \ldots, k_{M-1} \cdot X^{r_{M-1}}\}$ in $\mathcal{R}_Q$, where $r_i \in [N]$ and $k_i \in \mathbb{F}$. These monomials can be compactly represented by two vectors: $\boldsymbol{k} = (k_0, \ldots, k_{M-1})$, which stores the coefficients, and $\boldsymbol{r} = (r_0, \ldots, r_{M-1})$, which stores the exponents. However, to check the NTT computation for each monomial, we must represent it using its full coefficient vector, which also serves as the standard witness in the NTT relation $\mathcal{R}_{\text{NTT}}$. A simple observation is that the $i$-th monomial corresponds to a coefficient vector $\boldsymbol{c}_i$ with a single nonzero entry $k_i$ at position $r_i$. Stacking these vectors forms a sparse coefficient matrix $C^{N \times M} = (\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1})$, where each column contains exactly one nonzero element $k_i$, located at position $r_i$. This matrix corresponds precisely to the one considered in Section 3.1, with the sparse representation given by row $= \boldsymbol{r}$ and val $= \boldsymbol{k}$. As a result, evaluating $C^{N \times M}$ at a random point reduces to subclaims that query only $\boldsymbol{r}$ and $\boldsymbol{k}$.

The remaining step is to efficiently reduce the batched NTT argument to a single query on the sparse coefficient matrix $C^{N \times M}$. As discussed in Section 3.2, a batch of NTT instances can be proven via a single randomized NTT instance. Specifically, consider $M$ NTT operations on the coefficient vectors of monomials, denoted by $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1} \in \mathbb{F}^N$. Each NTT instance can be uniquely labeled by the tuple $(u, e_{\mathcal{Y}}, \boldsymbol{c}_{\mathcal{Y}})$ for $\mathcal{Y} \in [M]$, according to Definition 5. In Section 3.2, we introduced two approaches for randomizing batched NTT instances. Here, we adopt the second method, which treats the batch as a multilinear extension and checks the equality by evaluating it at a random point $r' \xleftarrow{\$} \mathbb{EF}^{\log M}$. More concretely, the randomized instance is defined as $(u, e', \boldsymbol{c}')$, where $e' = \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}(y, r') \cdot e_{\mathcal{Y}}$ and $\boldsymbol{c}' = \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}(y, r') \cdot \boldsymbol{c}_{\mathcal{Y}} \in \mathbb{F}^N$. Since each $\boldsymbol{c}_{\mathcal{Y}}$ has only one nonzero entry, and all scalars $\tilde{eq}(y, r')$ for $y \in \{0,1\}^{\log M}$ can be jointly precomputed in linear time, the randomized vector $\boldsymbol{c}'$ can be efficiently computed in $O(M)$. Then the prover and the verifier run the PIOP from Theorem 3.3 on this randomized NTT instance. This yields a subclaim of an evaluation query on $\tilde{\boldsymbol{c}}'(r)$ for a random $r \xleftarrow{\$} \mathbb{EF}^{\log N}$. Notably, this query corresponds exactly to an evaluation of the sparse coefficient matrix $C$ at the joint point $(r, r')$:

$$\tilde{\boldsymbol{c}}'(r) = \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}\left(y, r'\right) \tilde{\boldsymbol{c}}_{\mathcal{Y}}(r) \tag{6}$$

$$= \sum_{y \in \{0,1\}^{\log M}} \tilde{eq}\left(y, r'\right) \sum_{x \in \{0,1\}^{\log N}} \tilde{eq}\left(x, r\right) \boldsymbol{c}_{\mathcal{Y}}(x) \tag{7}$$

$$= \sum_{y \in \{0,1\}^{\log M}} \sum_{x \in \{0,1\}^{\log N}} \tilde{eq}\left(y, r'\right) \tilde{eq}\left(x, r\right) C(\mathcal{X}, \mathcal{Y}) \tag{8}$$

$$= \tilde{C}(r, r') \tag{9}$$

This means the query can be answered directly using the PIOP for sparse matrix evaluation described in Section 3.1. We now formally define our Mono-NTT relation $\mathcal{R}_{\text{Mono-NTT}}$, which captures $M$ NTT operations performed on monomials.

Def 9. *The Mono-NTT relation $\mathcal{R}_{\text{Mono-NTT}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, M, u), (\boldsymbol{k}, \boldsymbol{r}, \boldsymbol{e}))$$

*such that for each $i \in [M]$, the tuple $(\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathrm{R}}, N, u, e_i), (\boldsymbol{c}_i)) \in \mathcal{R}_{\text{NTT}}$, where $\boldsymbol{c}_i \in \mathbb{F}^N$ has exactly one nonzero entry $k_i \in \mathbb{F}$ at position $r_i \in \mathbb{Z}_N$.*

Theorem 4.2. *For every finite field $\mathbb{F}$ and its extension field $\mathbb{EF}$, and for all positive integers $N, M$, there exists a PIOP for $\mathcal{R}_{\text{Mono-NTT}}$ that achieves perfect completeness and soundness error $O(\log^2 N + M)/|\mathbb{EF}|$, with communication cost $O(\log^2 N + \log M)$. The prover runs in $O(M + N)$ time, and the verifier runs in $O(\log^2 N + \log M)$.*

We prove it with the Construction 6 in Appendix C.4.

**Algorithm 2** AccUpdate(acc, $\boldsymbol{d}$, Z, $B, \ell$)

**Input:** acc $\in \mathbb{RLWE}$, $\boldsymbol{d} \in \mathbb{F}^N$, Z $\in \mathbb{RGSW}$
**Input:** $B \in \mathbb{Z}$ and $\ell = \lceil \log_B Q \rceil$
**Output:** acc′ $\in \mathbb{RLWE}$ and trace

mult = $(\boldsymbol{d} \circ \text{acc}.\boldsymbol{a}, \boldsymbol{d} \circ \text{acc}.\boldsymbol{b}) \in \mathbb{RLWE}$
Imult = (INTT(mult.$\boldsymbol{a}$), INTT(mult.$\boldsymbol{b}$)) $\in \mathbb{RLWE}$
bits$[0..\ell]$ = Dec(Imult.$\boldsymbol{a}$; $B, \ell$)
bits$[\ell..2\ell]$ = Dec(Imult.$\boldsymbol{b}$; $B, \ell$)
**for** $i = 0$ to $2\ell$ **do**
  Nbits$[i]$ = NTT(bits$[i]$)
**end for**
prod = $\left(\sum_{i=0}^{2\ell-1} \text{Nbits}[i] \circ \text{Z}.\boldsymbol{a}_i, \sum_{i=0}^{2\ell-1} \text{Nbits}[i] \circ \text{Z}.\boldsymbol{b}_i\right)$
acc′ = acc + prod $\in \mathbb{RLWE}$
**return** acc′ and (mult, Imult, bits, Nbits, prod)

## 4.3 Accumulator Update

In this section, we propose a PIOP for proving the correctness of an accumulator update of the form: acc′ = acc + $\boldsymbol{d} \otimes$ acc $\diamond$ Z, which updates the accumulator state from acc to acc′. As described in Section 2.2, acc $\in \mathbb{RLWE}$ (and similarly for acc′) consists of two vectors acc.$\boldsymbol{a}$ and acc.$\boldsymbol{b}$ in $\mathbb{F}^N$. Each update consumes a vector $\boldsymbol{d} \in \mathbb{F}^N$ and an element Z $\in \mathbb{RGSW}$, where Z contains $4\ell$ vectors denoted by $\text{Z}.\boldsymbol{a}_0, \ldots, \text{Z}.\boldsymbol{a}_{2\ell-1}, \text{Z}.\boldsymbol{b}_0, \ldots, \text{Z}.\boldsymbol{b}_{2\ell-1} \in \mathbb{F}^N$. Here $\ell = \lceil \log_B Q \rceil$ denotes the number of gadgets in the gadget decomposition. This update is formalized in Algorithm 2, where the internal steps of the computation are recorded in trace.

In Algorithm 2, each accumulator update consists of $2\ell + 2$ NTT/INTT operations, two gadget decompositions, and $4\ell + 2$ Hadamard products. The basis $B$ is a tunable parameter: choosing a larger basis $B$ reduces both the number of NTT operations and Hadamard products, potentially resulting in a more efficient prover. The PIOP for proving the accumulator update relation $\mathcal{R}_{\text{acc}}$, defined below, follows naturally from the aforementioned PIOPs.

Def 10. *The accumulator update relation $\mathcal{R}_{\text{acc}}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\text{Z}, (\mathbb{F}, \mathbb{EF}, F^{\text{R}}, N, B, \ell), (\text{acc}, \boldsymbol{d}, \text{acc}', \text{trace}))$$

*where $\ell = \lceil \log_B Q \rceil$, Z $\in \mathbb{RGSW}$, acc, acc′ $\in \mathbb{RLWE}$, $\boldsymbol{d} \in \mathbb{F}^N$, and* (acc′, traces) = AccUpdate(acc, $\boldsymbol{d}$, Z, $B, \ell$). *The components of* trace *are defined in Algorithm 2.*

Theorem 4.3. *For every finite field $\mathbb{F}$ and its extension field $\mathbb{EF}$, and for all positive integers $N, B$, there exists a PIOP for the relation $\mathcal{R}_{\text{acc}}$ that achieves perfect completeness and soundness error $O(\ell N)/|\mathbb{EF}|$, with communication cost of $O(\log^2 N + \ell)$. The prover runs in $O(\ell N)$ time, and the verifier in $O(\log^2 N + \ell)$ time.*

We prove it with Construction 7 in Appendix C.5.

## 4.4 Modulus Switching

Modulus switching scales 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:

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

which rounds the fraction $\frac{a \cdot q}{Q}$ to the nearest integer, followed by a modular reduction modulo $q$. Notably, $\lfloor \frac{a \cdot q}{Q} \rceil$ may wrap around modulo $q$, potentially resulting in $b = 0$.

We consider modulus switching from $Q$ to $q$, under the assumption that $2q | Q - 1$. This assumption is satisfied under typical FHEW/TFHE parameter settings (see Appendix D.1 for further discussion). Define $k = \frac{Q-1}{2q}$, so that $Q - 1 = 2kq$ holds exactly. A key observation is that the modulus switching under this equality can be interpreted 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 $\mathbb{Z}_q$. To handle the special range $[0, k] \cup [Q - k, Q)$, we reformulate it as $\{k\} \cup [(2q-1) \cdot k + 1, (2q+1) \cdot k] \pmod Q$ using the equality $Q - 1 = 2kq$. By leveraging $b' \in \{1, \ldots, q\}$ such that $b \equiv b' \mod q$, we can represent modulus switching as a highly structured, unified, and parameterized mapping, as described in Theorem 4.4.

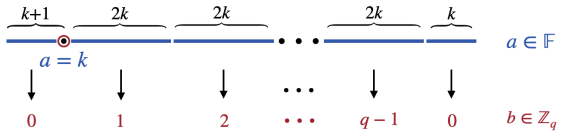


**Figure 3: Assuming $2q|Q-1$ with $k = \frac{Q-1}{2q}$, the modulus switching maps $a \in [0, k] \cup [Q-k, Q)$ to 0, and $a \in [(2b-1) \cdot k + 1, (2b+1) \cdot k]$ for each $b \in \{1, \ldots, q-1\}$ to $b \in \mathbb{Z}_q$.**

Theorem 4.4. *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 $a \in [(2b'-1) \cdot k + 1, (2b'+1) \cdot k] \subset \mathbb{F}$.*

The proof of Theorem 4.4 is provided in Appendix D.2.

Specifically, Theorem 4.4 extracts a dichotomy from the modulus switching relation $\mathcal{R}_{\text{switch}}$ (Definition 11). This dichotomy consists of two cases, each of which can be verified with an equality test. In the first case, where $a = k$ and $b = 0$, we prove correctness by checking that $p = \lambda_1 \cdot (a - k) + \lambda_2 \cdot b = 0$, where $\lambda_1 \xleftarrow{\$} \mathbb{EF}$ and $\lambda_2 \xleftarrow{\$} \mathbb{EF}$. In the second case, we prove that $c = a - (2b'-1) \cdot k - 1 \in \mathbb{F}$ lies within the range $[0, 2k)$ using the range check protocol based on gadget decomposition, as detailed in Section 3.3. By introducing a binary witness $w \in \{0, 1\}$ to indicate which case applies, we unify both conditions:

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

where $w = \mathbb{1}(a = k)$ can be computed efficiently. Based on this approach, we propose a PIOP for the modulus switching relation $\mathcal{R}_{\text{switch}}$, involving only Hadamard products and range checks.

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

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

*such that $\boldsymbol{a} \in \mathbb{F}_Q^N$ and $\boldsymbol{b} \in \mathbb{Z}_q^N$, under the assumption that $2q \mid Q - 1$. Here, $B \in \mathbb{Z}$ is a tunable basis. For each $i \in [N]$, we have $b_i = \lfloor \frac{a_i \cdot q}{Q} \rceil$.*

Theorem 4.5. *For every finite field $\mathbb{F}$, and its extension field $\mathbb{EF}$, and for all positive integers $N$ and $B$, there is a PIOP for the relation $\mathcal{R}_{\text{switch}}$ that achieves perfect completeness and soundness error $O(\ell N)/|\mathbb{EF}|$, with communication cost of $O(\log N + \ell)$, where $\ell = \left\lceil \log_B \frac{Q-1}{q} \right\rceil$. The prover runs in $O(\ell N)$ time, and the verifier in $O(\log N + \ell)$ time.*

We prove it with Construction 8 in Appendix C.6.

## 4.5 PIOP for FHE NAND

We propose a PIOP for the FHE NAND operation with bootstrapping, as described in Algorithm 3 and illustrated in Fig. 1. By invoking the PIOPs for all involved relations—each capturing the correctness of an essential step in the algorithm—we construct a PIOP for the overall NAND relation $\mathcal{R}_{\mathsf{NAND}}$ (Definition 12).

As described in Algorithm 3, it takes as input two $\mathbb{LWE}$ ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$, a constant vector $\boldsymbol{v}$, and $n$ $\mathbb{RGSW}$ ciphertexts $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1}$ serving as public bootstrapping keys. It outputs the resulting $\mathbb{LWE}$ ciphertext ct along with the computation traces. The traces include $n$ $\mathbb{RLWE}$ values and $n$ sub-traces $\{\mathsf{acc}[i], \mathsf{tc}[i]\}_{i=0}^{n-1}$, which record the accumulator states and their corresponding update trace at each update. The detailed components of traces and the sub-traces are defined in Algorithm 2 and Algorithm 3, respectively. The overall computation primarily consists of one LWE addition, $n+1$ NTT operations on monomials, $n$ accumulator updates, and $n+1$ modulus switching operations. The remaining lightweight operations can be captured using Hadamard relations.

Notably, by introducing an appropriate constant auxiliary vector, we can selectively apply different linear operations to individual entries of a vector. For example, using the auxiliary vector $\mathsf{aux}_\mathsf{b} = (0, \ldots, 0, 1) \in \mathbb{Z}_2^{n+1}$, the Hadamard identity $(q - \boldsymbol{a}) \cdot (1 - \mathsf{aux}_\mathsf{b}) + \boldsymbol{a} \cdot \mathsf{aux}_\mathsf{b} = 0$ holds if and only if $b_i = q - a_i$ for all $i \in [n]$ and $b_n = a_n$. This establishes the relation for verifying the correctness of Step 3 through 6. Similarly, Step 19 applies modulus switching selectively to the single entry $\mathsf{acc}.b_0$ using an auxiliary vector. Further details can be found in Construction 9 of Appendix C.7.

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

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

*where $\ell = \log_B Q$, $\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct} \in \mathbb{LWE}$, $\boldsymbol{v} \in \mathbb{F}^N$, $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1} \in \mathbb{RGSW}$ and* $(\mathsf{ct}, \mathsf{traces}) = \mathsf{HomNAND}(\mathsf{ct}_0, \mathsf{ct}_1, \boldsymbol{v}, \mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1}, B)$. *The components of* traces *are defined in Algorithm 3.*

Theorem 4.6. *For every finite field $\mathbb{F}$, and its extension field $\mathbb{EF}$, and for all positive integers $N, n, B$, there exists a PIOP for the relation $\mathcal{R}_{\mathsf{NAND}}$ that achieves perfect completeness and soundness error $O(\ell N n)/|\mathbb{EF}|$ with communication cost $O(\log^2 N + \ell n)$ where $\ell = \lceil \log_B Q \rceil$. The prover runs in $O(\ell N n)$ time, and the verifier in $O(\log^2 N + \ell n)$.*

We prove it with Construction 9 in Appendix C.7.

**Complexity and Security Analysis**. We focus on the arguments for proving the $n$ accumulator update relations, as they constitute the most computationally intensive part of the algorithm. In total, this part comprises $n \cdot (2\ell + 2)$ NTT/INTT operations, $2n$ gadget decompositions with $2n\ell$ decomposed gadgets, and $n \cdot (4\ell + 2)$ Hadamard products. Using the batching technique described in Section 3.2, the prover for the $n \cdot (2\ell + 2)$ NTT relations runs in $O(\ell N n)$ time, and the verifier in $O(\log^2 N + \ell n)$, where the $\ell n$ term accounts for evaluation queries on the original coefficient vectors. These queries are used to check the random linear combination between the randomized NTT relation and the original $O(\ell n)$ NTT relations. This batched NTT argument contributes a soundness error of $O(\log^2 N)/|\mathbb{EF}|$. The $2n$ gadget decompositions are actually reduced to a single batched lookup argument over $2n\ell$ vectors of length $N$, using the lookup table $\boldsymbol{t} = (0, \ldots, B-1)$. According to Theorem 3.4, the prover runs in $O(\ell N n)$, and both the verifier and the communication cost are $(\log N + n\ell)$. This sub-protocol incurs the dominant soundness error of $O(\ell N n)/|\mathbb{EF}|$. Since the protocol for proving Hadamard products is essentially a sum-check protocol, we can batch these sum-check sub-protocols with a random linear combination. As a result, the overall prover runtime remains $O(\ell N n)$, and both the verifier runtime and the communication cost are $O(\log^2 N + \ell n)$. The total soundness error of $\mathcal{R}_{\mathsf{NAND}}$ argument is dominated by the batched lookup argument for gadget decomposition and equals $O(\ell N n)/|\mathbb{EF}|$. The asymptotic complexity of all PIOPs for the internal FHE operations and the NAND operation is summarized in Table 4 of the Appendix.

**Algorithm 3** $\mathsf{HomNAND}(\mathsf{ct}_0, \mathsf{ct}_1, \boldsymbol{v}, \mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1})$

**Input:** $\mathsf{ct}_0, \mathsf{ct}_1 \in \mathbb{LWE}$, and $B \in \mathbb{Z}$
**Input:** $\boldsymbol{v} \in \mathbb{F}^N$ and $\mathsf{Z}_0, \ldots, \mathsf{Z}_{n-1} \in \mathbb{RGSW}$
**Output:** ct $\in \mathbb{LWE}$ and traces

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

# 5 HASTEBOOTS: SNARGS FOR FHE NAND

By using the cryptographic compiler in [20] to combine the PIOP for FHE NAND in Section 4.5 and the Brakedown PCS[41], we build HasteBoots, a succinct argument system that can efficiently generate proofs of FHE NAND with bootstrappings.

In our setting, given the bootstrapping key $\{\mathsf{Z}_i\}_{i=0}^{n-1}$ and the constant vector $\boldsymbol{v}$, both the prover and the verifier hold two public input $\mathbb{LWE}$ ciphertexts, $\mathsf{ct}_0$ and $\mathsf{ct}_1$ along with the output $\mathbb{LWE}$ ciphertext ct. The prover additionally holds the computation trace of the FHE NAND as the witness, enabling succinct verification time and communication. However, there is no need to protect the witness data, since the FHE circuit and the input (output) ciphertexts are all public. Under this setup, a SNARG system is sufficient without requiring knowledge extractors (i.e., SNARKs) nor requiring zero-knowledge; thus, there is no need for knowledge assumptions or masking polynomials, which improves the concrete efficiency.

## 5.1 Definitions

Formally speaking, let the index $\mathbb{i}$ contain bootstrapping keys 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 $\mathbb{w}$ = traces is computation traces defined in Algorithm 3. HasteBoots is a tuple of algorithms (PPGen, KeyGen, Prove, Verify). We give the formal definitions in Appendix F.

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

## 5.2 Polynomial Commitment Scheme

Polynomial commitment schemes (PCS) enable a prover to commit to a polynomial and later prove its evaluations at chosen points. They are widely used to implement oracles in PIOPs. This paper focuses on multilinear PCS, particularly Brakedown [41], which achieves a theoretically optimal linear-time prover with practical efficiency. For details on definitions and constructions, see [41].

We describe our optimization of packing small polynomials, as a single FHE NAND procedure involves 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 $\tilde{\boldsymbol{c}}_0(r), \ldots, \tilde{\boldsymbol{c}}_{M-1}(r)$ at the same point $r \in \mathbb{EF}^{\log N}$, the prover first computes $\tilde{\boldsymbol{c}}_i(r)$ for each $i \in [M]$ and sends these values to the verifier. These subclaims can be reduced to the evaluations on the matrix $\tilde{C}(r, r')$ via a random linear combination, where $r' \in \mathbb{EF}^{\log M}$ is sampled by the verifier. It is sufficient for the verifier to query the evaluation $\tilde{C}(r, r')$ at the joint point. The correctness follows the same idea in Eq. (6) when proving the reduction in the batched NTT on monomials in Section 4.2. This packing introduces soundness error of $\log M / |\mathbb{EF}|$ by the Schwartz-Zipple lemma.

| FHE Operation | | Proving Time | Verification Time | Proof Size |
|---|---|---|---|---|
| LWE Addition | | 17 ms | 6 ms | 11 MB |
| Bootstrapping | Mono-NTT | 151 ms | 36 ms | 35 MB |
| | Acc Update | 5.02 s | 219 ms | 107 MB |
| | Modulus Switching | 22 ms | 7 ms | 12 MB |
| **NAND** | | 5.13 s | 219 ms | 107 MB |

**Table 1:** Performance of HasteBoots on Apple M4. Results include both PIOP and Brakedown PCS costs.

# 6 IMPLEMENTATION AND EVALUATION

We implemented HasteBoots as a library in about 33K lines of Rust code. It includes PIOPs for all atomic and FHE operations, an optimized PCS based on Brakedown [41], and a framework for generating SNARGs tailored tp FHEW-like schemes. We plan to open-source our library.

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 ensure compatibility with our SNARG construction, setting $n = 1024$, $q = 1024$, $N = 1024$, and $Q = 2^{31} - 2^{27} + 1$. Our protocols operate over a 128-bit extension field of size $Q^D$ with $D = 4$ to ensure the soundness, while the underlying FHE instance is defined over a base field of size $Q$. For gadget decomposition, we set the gadget size to $B = 2^7$. In our implementation of the lookup protocol LogUp [43], we set the batch block size of 3 to minimize overhead. For Brakedown PCS [41], we set $\alpha = 0.1195$, $\beta = 0.0248$, and $r = 1.9$, where $\alpha$ controls the size of the subcode in each recursive call, $\beta$ defines the code distance, and $r$ adjusts the code rate. We also set a recursion stopping threshold of 10 to optimize performance.

**Performance**. We evaluate the performance of HasteBoots on an Apple M4, measuring the proving time, verification time and proof size for each FHE operation listed in Table 1. The asymptotic complexity for PIOPs and PCS are given in Appendix G.1. As shown in the results, the bootstrapping procedure is the most time-consuming part in the NAND circuit. Within the bootstrapping procedure, accumulator updates dominate the cost, accounting for 95% of the total time due to its complex workflow. Overall, HasteBoots achieves efficient proving time around 5.13 seconds, verification time of 219 ms, and a proof size of approximately 107 MB.

**Proving Time v.s. Proof Size**. To better understand the performance trade-offs in our SNARG construction, we isolate the contributions of the PCS and PIOP components in Table 2. The PCS cost consists of the polynomial commitment time and the time to generate evaluation proofs for large committed polynomials, as described in Section 5.2. With the Brakedown PCS, the prover spends around 25% of the total proving time on PCS, while PCS accounts for 95% of the verifier's total time and proof size.

To explore the trade-off between proof size and proving time, we replace the Brakedown PCS with BaseFold [68], using its implementation in Jolt-b [25]. As shown in Table 2, this substitution significantly reduces both the verification time and proof size, albeit at the cost of increased proving time. Nonetheless, even with this trade-off, our total proving time remains faster than the state-of-the-art result–around half an hour–reported in [61].

| | | Proving Time | Verification Time | Proof Size |
|---|---|---|---|---|
| NAND w/ Brakedown | PCS | 1.39 s | 212.5 ms | 104 MB |
| | PIOP | 3.75 s | 6.9 ms | 3 MB |
| | SNARG | 5.13 s | 219.4 ms | 107 MB |
| NAND w/ BaseFold | PCS | 63.74 s | 4 ms | 8 MB |
| | PIOP | 3.75 s | 6.9 ms | 3 MB |
| | SNARG | 67.49 s | 10.9 ms | 11 MB |

**Table 2:** Comparison using Brakedown PCS vs. BaseFold PCS.

| **Proving Time** | **RISC0** [59] | **SP1** [58] | [61] | **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 | – | – | – | **5 s** |

**Table 3:** Proving time for a single bootstrapping procedure across various devices.

**Comparison with Prior Work**. We compared the performance of HasteBoots with previous work on proving the FHE bootstrapping, as summarized in Table 3. zkVM-based approaches such as RISC0 [59] and SP1 [58], require roughly 3 days and 1 day of proving time, respectively, primarily due to performance losses when compiling complex FHE programs. Thibault and Walter [61] design a SNARK-friendly arithmetic circuit for TFHE bootstrapping procedure and prove it with Plonk [33], reducing proving time to around half an hour. However, as a general-purpose SNARK, Plonk still suffers from some inefficiency when applied FHE-specific operations. In contrast, avoiding proving the full arithmetic circuit, HasteBoots directly prove the relation between the inputs and the outputs of these FHE operations, enabling sub-10 second proof generation.

Concretely, [61] proves an explicit NTT circuit of size $O(N \log N)$, resulting in at least quasi-linear prover time. Our approach, by contrast, proves the NTT relation directly, avoiding circuit encoding and achieving an optimal linear-time prover. Moreover, [61] employs a decomposition method to compute multiplication by a monomial, leading to another circuit of size $O(N \log N)$, while our protocol achieves a linear-time protocol for batched NTT operation on monomials. Finally, [61] introduces additional noise to FHE ciphertexts in modulus switching, which is avoided in our protocol.

[61] achieves a verification time of approximately 8 ms and a proof size of around 200 KB, whereas HasteBoots requires roughly 219 ms and 107 MB. Notably, both the proof size and verification time of HasteBoots can be further optimized by adopting recursion techniques and replacing Brakedown with a more efficient PCS.

## 7 RELATED WORKS

**Fully Homomorphic Encryption**. There are two main categories of fully homomorphic encryption schemes. The first category is CKKS[19], BFV[9, 29], and BGV[10], 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[28], TFHE[21], and variants[6, 24, 46, 49]. They don't support batching, but they can perform very fast bootstrapping. We focused on the FHEW scheme with GINX[34] 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 [16, 17, 22, 47, 63, 69] 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 [4, 7, 18, 20, 33, 41, 42, 56, 66] are designed for arbitrary computations. Zero-knowledge Virtual Machine (zkVM) [58, 59] 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**. While recent research has advanced theoretical constructions for verifiable FHE, practical and efficient solutions remain elusive. Early work [5, 30, 31, 35, 37, 39, 62] supports only basic FHE operations such as LWE additions and omits bootstrapping, thus falling short of enabling fully homomorphism. [12, 52] show that verifiable FHE is possible with SNARKs. [3] provides a theoretical construction by encoding complete FHE using standard R1CS representations, but this method does not natively support the field arithmetic of FHE. To address this, [45] 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 proof techniques, which are not tailored for FHE. In contrast, our work directly proves FHE's core operations without converting them into generic circuits.

A recent line of work focuses on constructing concrete SNARKs for verifiable FHE. One approach is to use high-level languages, such as Rust, to implement FHE and then compile it into proofs using zkVM frameworks like R1SC0 and SP1 [58, 59]. However, this method suffers from extremely low efficiency, with proof generation taking several days. A more refined approach by Thibault and Walter [61] designs a SNARK-friendly arithmetic circuit for TFHE bootstrapping procedure and proves it using Plonk [33], reducing proving time to around half an hour. Yet, general-purpose SNARKs are not optimized for the specialized structure of FHE operations, limiting their performance in this context. In contrast, our work introduces a concrete succinct argument specifically tailored for FHE, enabling proof generation in seconds and achieving significant improvements in both efficiency and practical usability.

Another line of work [2, 14, 15, 36] 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 [11, 44], where the result on the chain needs to be verified by anyone. Second, these approaches are vulnerable to active attacks [23] on FHE.

A potential alternative approach toward verifiable FHE leverages the interactive proofs, such as Vector Oblivious Linear Evaluations (VOLE) [8, 26, 27, 64, 65, 67]. However, VOLE is mainly used to achieve zero-knowledge rather than succinctness, leading to verifier cost at least linear in the prover. However, the verifiable FHE computations are public, so succinctness is more important.

## ACKNOWLEDGMENTS

Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 92270201 and 62125204), and by the New Corner-stone Science Foundation through the XPLORER PRIZE. Xiang Xie is supported by the project "Research on Verifiable Fully Homomorphic Encryption Algorithms" (Project No. 23511101300), a key technology research initiative on blockchain under the 2023 Shanghai Science and Technology Innovation Action Plan. Yuncong Hu is supported by the National Natural Science Foundation of China, the Science and Technology Commission of Shanghai Municipality, the Shanghai Science and Technology Program (Grant No. 24BC3200200), the ExploreX project of SJTU, and gifts/awards from BIANJIE.AI, Polyhedra, and Xiaomi.

## REFERENCES

[1] Shweta Agrawal and Dongdai Lin (Eds.). 2022. *ASIACRYPT 2022, Part II*. LNCS, Vol. 13792. Springer, Cham.

[2] Diego F Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. 2023. Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. *ePrint* (2023).

[3] Shahla Atapoor, Karim Baghery, Hilder VL Pereira, and Jannik Spiessens. 2024. Verifiable FHE via Lattice-based SNARKs. *ePrint* (2024).

[4] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP 2018 (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl, 14:1–14:17. https://doi.org/10.4230/LIPIcs.ICALP.2018.14

[5] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. 2021. Flexible and efficient verifiable computation on encrypted data. In *PKC*.

[6] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. 2022. FINAL: Faster FHE Instantiated with NTRU and LWE, See [1], 188–215. https://doi.org/10.1007/978-3-031-22966-4_7

[7] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orru. 2022. Gemini: Elastic SNARKs for diverse environments. In *EUROCRYPT*.

[8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 896–912.

[9] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 309–325. https://doi.org/10.1145/2090236.2090262

[11] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards privacy in a smart contract world. In *FC*.

[12] Sébastien Canard, Caroline Fontaine, Duong Hieu Phan, David Pointcheval, Marc Renard, and Renaud Sirdey. 2024. Relations among new CCA security notions for approximate FHE. *ePrint* (2024).

[13] Anne Canteaut and Yuval Ishai (Eds.). 2020. *EUROCRYPT 2020, Part I*. LNCS, Vol. 12105. Springer, Cham.

[14] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2022. Verifiable encodings for secure homomorphic analytics. *arXiv:2207.14071* (2022).

[15] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2023. Poster: Verifiable Encodings for Maliciously-Secure Homomorphic Encryption Evaluation. In *CCS*.

[16] Bhuvnesh Chaturvedi, Anirban Chakraborty, Ayantika Chatterjee, and Debdeep Mukhopadhyay. 2023. vr$^2$FHE- Securing FHE from Reaction-based Key Recovery Attacks. Cryptology ePrint Archive, Report 2023/561. https://eprint.iacr.org/2023/561

[17] Bhuvnesh Chaturvedi, Anirban Chakraborty, Ayantika Chatterjee, and Debdeep Mukhopadhyay. 2024. "Ask and Thou Shall Receive": Reaction-Based Full Key Recovery Attacks on FHE. In *ESORICS 2024, Part IV (LNCS, Vol. 14985)*, Joaquin Garcia-Alfaro, Rafał Kozik, Michał Choraś, and Sokratis Katsikas (Eds.). Springer, Cham, 457–477. https://doi.org/10.1007/978-3-031-70903-6_23

[18] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *EUROCRYPT 2023, Part II (LNCS, Vol. 14005)*, Carmit Hazay and Martijn Stam (Eds.). Springer, Cham, 499–530. https://doi.org/10.1007/978-3-031-30617-4_17

[19] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Cham, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS, See [13], 738–768. https://doi.org/10.1007/978-3-030-45721-1_26

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (Jan. 2020), 34–91. https://doi.org/10.1007/s00145-019-09319-x

[22] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. 2016. Attacking FHE-based applications by software fault injections. Cryptology ePrint Archive, Report 2016/1164. https://eprint.iacr.org/2016/1164

[23] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. 2016. Attacking FHE-based applications by software fault injections. *ePrint* (2016).

[24] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2021. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *ASIACRYPT 2021, Part III (LNCS, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 670–699. https://doi.org/10.1007/978-3-030-92078-4_23

[25] Cysic Labs. 2024. jolt-b: A High-Performance Implementation of the Jolt Protocol. https://github.com/cysic-labs/jolt-b. Commit abc1234, accessed April 13, 2025.

[26] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. 2022. Improving line-point zero knowledge: Two multiplications for the price of one. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 829–841.

[27] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. 2020. Line-point zero knowledge and its applications. *Cryptology ePrint Archive* (2020).

[28] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Berlin, Heidelberg, 617–640. https://doi.org/10.1007/978-3-662-46800-5_24

[29] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144

[30] Dario Fiore, Rosario Gennaro, and Valerio Pastro. 2014. Efficiently verifiable computation on encrypted data. In *CCS*.

[31] Dario Fiore, Anca Nitulescu, and David Pointcheval. 2020. Boosting verifiable computation on encrypted data. In *PKC*.

[32] Marc Fischlin and Jean-Sébastien Coron (Eds.). 2016. *EUROCRYPT 2016, Part II*. LNCS, Vol. 9666. Springer, Berlin, Heidelberg.

[33] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953

[34] Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. 2016. Structural Lattice Reduction: Generalized Worst-Case to Average-Case Reductions and Homomorphic Cryptosystems, See [32], 528–558. https://doi.org/10.1007/978-3-662-49896-5_19

[35] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. 2023. Rinocchio: SNARKs for ring arithmetic. *JoC* (2023).

[36] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. 2024. How to prove statements obliviously?. In *CRYPTO*.

[37] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*.

[38] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440

[39] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. 2013. How to run turing machines on encrypted data. In *CRYPTO*.

[40] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. 2021. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. Cryptology ePrint Archive, Paper 2021/1043. https://eprint.iacr.org/2021/1043

[41] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. 2023. Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS. In *CRYPTO 2023, Part II (LNCS, Vol. 14082)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, Cham, 193–226. https://doi.org/10.1007/978-3-031-38545-2_7

[42] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments, See [32], 305–326. https://doi.org/10.1007/978-3-662-49896-5_11

[43] Ulrich Haböck. 2022. Multivariate lookups based on logarithmic derivatives. *Cryptology ePrint Archive* (2022).

[44] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. 2020. Ghostor: Toward a Secure {Data-Sharing} System from Decentralized Trust. In *NSDI*.

[45] Mi-Ying Miryam Huang, Baiyu Li, Xinyu Mao, and Jiapeng Zhang. 2024. Fully Homomorphic Encryption with Efficient Public Verification. *ePrint* (2024).

[46] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. 2023. Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption. In *EUROCRYPT 2023, Part III (LNCS, Vol. 14006)*, Carmit Hazay and Martijn Stam

(Eds.). Springer, Cham, 227–256. https://doi.org/10.1007/978-3-031-30620-4_8
[47] Zengpeng Li, Steven D. Galbraith, and Chunguang Ma. 2016. Preventing Adaptive Key Recovery Attacks on the GSW Levelled Homomorphic Encryption Scheme. In *ProvSec 2016 (LNCS, Vol. 10005)*, Liqun Chen and Jinguang Han (Eds.). Springer, Cham, 373–383. https://doi.org/10.1007/978-3-319-47422-9_22
[48] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.
[49] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2022. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping, See [1], 130–160. https://doi.org/10.1007/978-3-031-22966-4_5
[50] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)* 39, 4 (1992), 859–868.
[51] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Berlin, Heidelberg, 1–23. https://doi.org/10.1007/978-3-642-13190-5_1
[52] Mark Manulis and Jérôme Nguyen. 2024. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. In *EUROCRYPT*.
[53] Daniele Micciancio and Yuriy Polyakov. 2020. Bootstrapping in FHEW-like Cryptosystems. Cryptology ePrint Archive, Report 2020/086. https://eprint.iacr.org/2020/086
[54] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. https://doi.org/10.1145/1060590.1060603
[55] Ardianto Satriawan, Rella Mareta, and Hanho Lee. 2024. A Complete Beginner Guide to the Number Theoretic Transform (NTT). *Cryptology ePrint Archive* (2024).
[56] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.
[57] Srinath Setty, Justin Thaler, and Riad Wahby. 2024. Unlocking the lookup singularity with Lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 180–209.
[58] The SuccinctLabs. [n. d.]. The fastest, most feature-complete zkVM for developers. https://github.com/succinctlabs/sp1.
[59] The RISC Zero Team. [n. d.]. Universal Zero Knowledge. https://risczero.com/.
[60] Justin Thaler. 2013. Time-optimal interactive proofs for circuit evaluation. In *Annual Cryptology Conference*. Springer, 71–89.
[61] Louis Tremblay Thibault and Michael Walter. 2024. Towards Verifiable FHE in Practice: Proving Correct Execution of TFHE's Bootstrapping using plonky2. *ePrint* (2024).
[62] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. 2023. Verifiable fully homomorphic encryption. *arXiv:2301.07041* (2023).
[63] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. 2023. Verifiable Fully Homomorphic Encryption. arXiv:2301.07041 [cs.CR] https://arxiv.org/abs/2301.07041
[64] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1074–1091.
[65] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. 2022. AntMan: Interactive zero-knowledge proofs with sublinear communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2901–2914.
[66] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. 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, 733–764.
[67] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2986–3001.
[68] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2024. BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes. In *Annual International Cryptology Conference*. Springer, 138–169.
[69] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. 2012. Reaction Attack on Outsourced Computing with Fully Homomorphic Encryption Schemes. In *ICISC 11 (LNCS, Vol. 7259)*, Howon Kim (Ed.). Springer, Berlin, Heidelberg, 419–436. https://doi.org/10.1007/978-3-642-31912-9_28

# A FHE NAND WITH BOOTSTRAPPING

Given two LWE ciphertexts $\mathrm{ct}_0, \mathrm{ct}_1$ encrypt two bits with binary secret key $\boldsymbol{s} = (s_0, ..., s_{n-1})$. We first add these two ciphertexts into a new ciphertext $\mathrm{ct} = (\boldsymbol{a}, b) \in \mathbb{Z}_q^{n+1}$ 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 $Y^{-a_0}, \ldots, Y^{-a_{n-1}}$ and $Y^b$. 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* step.

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 $\mathrm{acc}[0] = (0, \boldsymbol{v} \cdot Y^b)$ is iteratively updated as $\mathrm{acc}[i+1] = \mathrm{acc}[i] + Y^{-a_i} \otimes \mathrm{acc}[i] \diamond \mathsf{Z}_i$ 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: $x \in \mathbb{Z}_Q \to \lfloor \frac{x \cdot q}{Q} \rceil \bmod q$.

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.

# B ADDITIONAL DETAILS ON USEFUL PIOP

## B.1 Sum-check Protocol

We describe a seminal interactive proof used in our work, known as the sumcheck protocol, proposed by Lund et al.[50]. Suppose there is an $\mu$-variate low-degree polynomial $\mathcal{G} : \mathbb{F}^\mu \to \mathbb{F}$ where the degree of each variable in $\mathcal{G}$ is at most $\ell$. 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 sumcheck protocol, $\mathcal{V}$ interacts with $\mathcal{P}$ over $\mu$ 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 $r \in \{0,1\}^*$, $\Pr\left[\langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T) = 1\right] = 1$.
- **Soundness.** If $T \neq \sum_{x \in \{0,1\}^\mu} \mathcal{G}(x)$, then for any $\mathcal{P}^*$ and for all $r \in \{0,1\}^*$, $\Pr\left[\langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle (\mu, \ell, T) = 1\right] \le \ell \cdot \mu / |\mathbb{EF}|$.
- **Succinctness.** The proof size is $O(\mu \cdot \ell)$ elements of $\mathbb{EF}$.

## B.2 Hadamard Products

We provide a public-coin PIOP in Construction 1 for the Hadamard relation $\mathcal{R}_{\mathsf{sparse}}$, which is basically proven by a sum-check protocol. Specifically, $\sum_{i=0}^{M-1} \boldsymbol{a}_i(x) \circ \boldsymbol{b}_i(x) = \boldsymbol{c}(x)$ for all $x \in \{0,1\}^{\log N}$ holds

| Protocols | Proving Time | Verification Time | Soundness Error | Round | Communication |
|---|---|---|---|---|---|
| | | *Atomic Operations* | | | |
| Hadamard Protocol | $O(M_1N)$ | $O(\log N + M_1)$ | $O(\log N)$ | $O(\log N)$ | $O(\log N + M_1)$ |
| Batched Lookups | $O(M_2N)$ | $O(\log N + M_2)$ | $O(M_2N)$ | $O(\log N)$ | $O(\log N + M_2)$ |
| **Sparse Matrix Evaluation** | $O(M_3)$ | $O(\log M_3)$ | $O(M_3)$ | $O(\log M_3)$ | $O(\log M_3)$ |
| **NTT** | $O(N)$ | $O(\log^2 N)$ | $O(\log^2 N)$ | $O(\log^2 N)$ | $O(\log^2 N)$ |
| **Gadget Decomposition** | $O(L_1N)$ | $O(\log N + L_1)$ | $O(L_1N)$ | $O(\log N)$ | $O(\log N + L_1)$ |
| | | *FHE Operations* | | | |
| **LWE Addition** | $O(n)$ | $O(\log n)$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ |
| **Mono-NTT** | $O(n + N)$ | $O(\log^2 N + \log n)$ | $O(\log^2 N + n)$ | $O(\log^2 N + \log n)$ | $O(\log^2 N + \log n)$ |
| **Accumulator Update** | $O(L_1N)$ | $O(\log^2 N + L_1)$ | $O(L_1N)$ | $O(\log^2 N)$ | $O(\log^2 N + L_1)$ |
| **Modulus Switching** | $O(L_2n)$ | $O(\log n + L_2)$ | $O(L_2n)$ | $O(\log n)$ | $O(\log n + L_2)$ |
| **FHE NAND** | $O(L_1Nn)$ | $O(\log^2 N + L_1n)$ | $O(L_1Nn)$ | $O(\log^2 N + \log n)$ | $O(\log^2 N + L_1n)$ |

**Table 4: Asymptotic complexity of PIOPs used throughout our paper. Proving and verification times, and communication costs are measured in terms of (extension) field operations. Soundness error is given in multiples of $1/|\mathbb{EF}|$. Notation: Let $Q$ be the modulus of $\mathbb{RLWE}$ ciphertexts and $q$ the modulus of $\mathbb{LWE}$ ciphertexts such that $2q \mid (Q-1)$. Let $B$ be a tunable decomposition base, and define $L_1 = \lceil \log_B Q \rceil$ and $L_2 = \lceil \log_B(\frac{Q-1}{q}) \rceil$. Other parameters include $N$ (vector length), $n$ (dimension of the $\mathbb{LWE}$ vectors), $M_1$ (number of Hadamard products within the sum), $M_2$ (number of lookup vectors), $M_3$ (column count/sparsity of the sparse matrix).**

if and only if the multivariate polynomial $\mathcal{F}(t) = \sum_{x\in\{0,1\}^{\log N}} \tilde{eq}(x,t)\cdot(\sum_{i=0}^{M-1} \boldsymbol{a}_i(x)\cdot\boldsymbol{b}_i(x) - \boldsymbol{c}(x))$ is identically zero. Hence, it suffices to check whether $\mathcal{F}(\mu) = 0$ at a randomly sampled $\mu \xleftarrow{\$} \mathbb{EF}^{\log N}$. By the Schwartz-Zippel lemma, this introduces a soundness error of at most $\log N/|\mathbb{EF}|$. Combined with the soundness error of the sum-check protocol, the overall soundness error of this PIOP is $O(\log N/|\mathbb{EF}|)$. Completeness follows from the perfect completeness of the sum-check protocol and the fact that if the Hadamard relation holds, the multivariate polynomial is identically zero, implying $\mathcal{F}(u) = T = \sum_{x\in\{0,1\}^{\log N}} \mathcal{G}(x) = 0$.

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

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

The prover runs in linear time with $O(MN)$ computation while the verifier's computation is $O(\log N + M)$ where $M$ comes from query complexity. Since the degree of $\mathcal{G}$ in Step 2 is constant and the round complexity of the sum-check protocol is $O(\log N)$, the total communication cost is $O(\log N + M)$, where $M$ accounts for the number of evaluations the prover sends in Step 4.

## B.3 Lookups

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 [43]. 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 $\boldsymbol{t} \in \mathbb{F}^{N'}$, lookup problem can be written as $\{a_i : i = 1,\dots,N\} \subseteq \{t_j : j = 1,\dots,N'\}$ as sets, with multiples of values removed. For simplicity, we assume $N = N'$. According to the Lemma 5 of [43], 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 $\mathrm{mp}(\boldsymbol{a}, z)$ 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 $\boldsymbol{m}_i = \frac{\mathrm{mp}(\boldsymbol{a},t_i)}{\mathrm{mp}(\boldsymbol{t},t_i)}$ in probablisitic polynomial time.

We consider the batch lookup arguments proposed in [43]. Let $\boldsymbol{f}_0,\dots,\boldsymbol{f}_{M-1}$ and $\boldsymbol{t}$ be multivariate functions defined over the Boolean hypercube, i.e., $\{0,1\}^{\log N} \to \mathbb{F}$. By Lemma 5 in [43], 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 $\boldsymbol{m}(x) = \frac{\sum_{i=0}^{M-1}\mathrm{mp}(\boldsymbol{f}_i,\boldsymbol{t}(x))}{\mathrm{mp}(\boldsymbol{t},\boldsymbol{t}(x))}$.

Hence, we obtain a PIOP as described in Construction 2 for the relation $\mathcal{R}_{\mathsf{lookup}}$ with perfect completeness and soundness error $O(MN)/|\mathbb{EF}|$. The prover runs in $O(MN)$ time, while the verifier's runtime is $O(\log N+M)$. The total communication cost is $O(\log N+M)$. The complexity and security analyses follow from [43].

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

1. *$\mathcal{P}$ computes the multiplicity polynomial $\boldsymbol{m}(x) = \frac{\sum_{i=0}^{M-1} \mathrm{mp}(\boldsymbol{f}_i, \boldsymbol{t}(x))}{\mathrm{mp}(\boldsymbol{t}, \boldsymbol{t}(x))}$ and sends the oracle message to $\mathcal{V}$.*
2. *$\mathcal{V}$ samples a random challenge $\rho \xleftarrow{\$} \mathbb{EF}$ and sends it to $\mathcal{P}$.*
3. *$\mathcal{P}$ 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 $\boldsymbol{\varphi}_M(x) = \rho + \boldsymbol{t}(x)$. $\mathcal{P}$ sends the oracle messages $\boldsymbol{h}_0, \ldots, \boldsymbol{h}_M$ to $\mathcal{V}$.*
4. *Let $\mathcal{G} = \sum_{i=0}^{M} \boldsymbol{h}_i(x), T = 0, \mu = \log N, \ell = 1$.*
5. *$\mathcal{P}$ and $\mathcal{V}$ run the sumcheck protocol $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle$ $(\mu, \ell, T)$ where $r \in \mathbb{EF}^{\mu}$.*
6. *$\mathcal{V}$ makes the evaluation queries for $i \in [M+1]$: $e^{\boldsymbol{h}_i} = \boldsymbol{h}_i(r)$, and check that $\sum_{i=0}^{M} e^{h} i = e$.*
7. *For $i \in [M+1]$, $\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol in parallel with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{h}_i, \boldsymbol{\varphi}_i, \boldsymbol{m}_i))$*

Moreover, an indexed lookup construction can be readily obtained by applying a public-coin hash function to the index and the table value, introducing only a negligible soundness error. The PIOP for the relation $\mathcal{R}_{\mathsf{id-lookup}}$ is omitted here and has perfect completeness and a soundness error $O(N)/|\mathbb{EF}|$. The prover runs in $O(N)$ time, while the verifier's runtime is $O(\log N)$.

# C SECURITY PROOF

## C.1 Proof of Theorem 3.2

To prove Theorem 3.2, we first provide Construction 3, which defines a public-coin PIOP for sparse matrix evaluation relation $\mathcal{R}_{\mathsf{sparse}}$. This construction is a PIOP derived from [57] with perfect completeness and soundness error of $O(M)/|\mathbb{EF}|$, where the memory-checking is simply replaced with an indexed lookup PIOP.

Construction 3. *We construct a PIOP for the relation $\mathcal{R}_{\mathsf{sparse}}$. $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, the public input $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, M, e, r_x, r_y)$, and the witness $\mathbb{w} = (\mathsf{val}, \mathsf{row})$; the verifier $\mathcal{V}$ takes as input the index $\mathbb{i}$ and the instance $\mathbb{x}$.*

1. *$\mathcal{P}$ computes $E_{r_x}(k) = \tilde{eq}\,(\text{to-bits}(\mathsf{row}(k)), r_x)$ for $k \in \{0,1\}^{\log M}$.*
2. *Let $\mathcal{G}(k) = \mathsf{val}(k) \cdot E_{r_x}(k) \cdot \tilde{eq}(k, r_y), T = e, \mu = \log M, \ell = 3$.*
3. *$\mathcal{P}$ and $\mathcal{V}$ run the sumcheck protocol $e \leftarrow \langle \mathcal{P}(\mathcal{G}), \mathcal{V}(r) \rangle$ $(\mu, \ell, T)$ where $r \in \mathbb{EF}^{\log M}$.*
4. *$\mathcal{V}$ makes the evaluation queries: $e^{\mathsf{val}} := \mathsf{val}(r), e^{E_{r_x}} := E_{r_x}(r)$, and checks that $e^{\mathsf{val}} \cdot e^{E_{r_x}} \cdot \tilde{eq}(r, r_y) = e$.*
5. *Let $\boldsymbol{t}(x) = \tilde{eq}(x, r_x)$ for $x \in \{0,1\}^{\log N}$ and $\mathsf{id}(k) = \mathsf{row}(k)$ for $k \in \{0,1\}^{\log M}$. For simplicity, assuming $N \le M$, let $\boldsymbol{t'}(k)$ be the padded table of $\boldsymbol{t}(x)$ where $k \in \{0,1\}^{\log M}$.*
6. *$\mathcal{P}$ and $\mathcal{V}$ check that $E_{r_x}$ is well-formed using indexed lookup with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, M, \boldsymbol{t'}), (\mathsf{id}, E_{r_x}))$*

Step 1 and 5 can be computed in linear time using a dynamic programming algorithm to compute $\tilde{eq}(\cdot, r_x)$ over the hypercube. With a linear-time prover for the sum-check protocol and a linear-time prover for the indexed lookup protocol, the prover runs in $O(M)$ time, and the verifier's runtime is $O(\log M)$. The total communication cost is $O(\log M)$.

## C.2 Proof of Theorem 3.4

To prove Theorem 3.4, we first provide Construction 4, which defines a public-coin PIOP for the gadget decomposition relation $\mathcal{R}_{\mathsf{dec}}$ with perfect completeness and a soundness error of $O(\ell N)/|\mathbb{EF}|$. The core of this construction reduces to batched lookup arguments for proving the range of $\ell$ gadgets with a soundness error $O(\ell N)/|\mathbb{EF}|$. To verify equality of $\sum_{i=0}^{\ell-1} B^i \cdot \tilde{a}_i(x) = \tilde{a}(x)$, it suffices to evaluate at a random point since both sides are multilinear in $x$, introducing a soundness error of at most $\log N/\mathbb{EF}$.

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

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

The prover's runtime is dominated by the batched lookup reductions, resulting in $O(\ell N)$ extension field operations. The verifier's runtime is $O(\log N + \ell)$.

## C.3 Proof of Theorem 4.1

To prove Theorem 4.1, we first introduce Construction 5, which defines a public-coin PIOP for LWE addition relation $\mathcal{R}_{\mathsf{add}}$ with perfect completeness and a soundness error of $O(N)/|\mathbb{EF}|$. The soundness error can be easily analyzed through the following construction.

Construction 5. *We construct a PIOP for the relation $\mathcal{R}_{\mathsf{add}}$. The prover $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, the public input $\mathbb{x} = (q, N, \boldsymbol{a}, \boldsymbol{b})$, and the witness $\mathbb{w} = \boldsymbol{c}$.*

1. *Define a table $\boldsymbol{t} = (0, \ldots, q-1, 0, \ldots, 0)$, and $\mathcal{P}$ and $\mathcal{V}$ run the lookup protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, \boldsymbol{t}, 1), \boldsymbol{c})$*
2. *$\mathcal{P}$ computes for all $x \in \{0,1\}^{\log N} : \boldsymbol{k}(x) = \mathbb{1}(\boldsymbol{a}(x) + \boldsymbol{b}(x) \ge q)$ and sends it to $\mathcal{V}$ as an oracle.*
3. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\{\boldsymbol{k}\}, \{1 - \boldsymbol{k}\}, 0))$.*
4. *$\mathcal{V}$ samples $r \xleftarrow{\$} \mathbb{EF}^{\log N}$ and makes the queries : $e^{\boldsymbol{a}} := \boldsymbol{a}(r), \quad e^{\boldsymbol{b}} := \boldsymbol{b}(r), \quad e^{\boldsymbol{c}} := \boldsymbol{c}(r), \quad e^{\boldsymbol{k}} := \boldsymbol{k}(r)$ and checks that $e^{\boldsymbol{a}} + e^{\boldsymbol{b}} = e^{\boldsymbol{c}} + q \cdot e^{\boldsymbol{k}}$.*

Since Step 2 requires only linear time, and both the lookup and Hadamard protocols have linear-time provers, the prover's overall runtime is $O(N)$. The verifier's runtime is $O(\log N)$.

## C.4 Proof of Theorem 4.2

To prove Theorem 4.2, we first provide Construction 6, which defines a public-coin PIOP for the Mono-NTT relation $\mathcal{R}_{\mathsf{Mono-NTT}}$ with perfect completeness and soundness error of $O(\log^2 N+M)/|\mathbb{EF}|$. Correctness follows from the linearity of NTT operations and the fact of the query on $\boldsymbol{c'}$ corresponds exactly to an evaluation of the sparse coefficient matrix at the joint point $\tilde{\boldsymbol{c}}'(r)$, as shown in Eq. (6).

As for the soundness, if any of the $M$ NTT instances is incorrect, the probability that the corresponding randomized NTT instance appears correct is at most $\log M/|\mathbb{EF}|$, by the Schwartz-Zippel lemma. Applying a union bound to account for this error and the soundness error of NTT protocol to check the single randomized NTT instance, as well as the sparse matrix evaluation protocol, we conclude the overall soundness error is bounded by $O(M + \log^2 N)/|\mathbb{EF}|$

Construction 6. *We construct a PIOP for the relation $\mathcal{R}_{\mathsf{Mono-NTT}}$. $\mathcal{P}$ takes as input an index $\mathbb{i} = \perp$, the public input $\mathbb{x} = (\mathbb{F}, \mathbb{EF}, N, M, u)$, and the witness $\mathbb{w} = (\boldsymbol{k}, \boldsymbol{r}, \boldsymbol{e})$.*

1. *$\mathcal{V}$ samples $r' \xleftarrow{\$} \mathbb{EF}^{\log M}$ and queries: $e' := \boldsymbol{e}(r')$*
2. *$\mathcal{P}$ computes the coefficient vector $\boldsymbol{c}' \in \mathbb{EF}^N$ of the randomized polynomial $\sum_{y\in\{0,1\}^{\log M}} \tilde{eq}(y, r') \cdot (k_{\boldsymbol{y}} \cdot X^{r_{\boldsymbol{y}}})$.*
3. *$\mathcal{P}$ and $\mathcal{V}$ run the NTT protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, u, e'), (\boldsymbol{c}'))$ and return the subclaim $d \stackrel{?}{=} \tilde{\boldsymbol{c}}'(r)$ for a random point $r \in \mathbb{EF}^{\log N}$.*
4. *$\mathcal{P}$ and $\mathcal{V}$ run the sparse matrix evaluation protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, M, d, r, r'), (\boldsymbol{k}, \boldsymbol{r}))$.*

In Step 2, all values $\tilde{eq}(y, r')$ for $y \in \{0,1\}^{\log M}$ can be jointly precomputed in $O(M)$ time using a dynamic programming algorithm. Since each term in the sum is a monomial with only one non-zero coefficient, the coefficient vector $\boldsymbol{c}'$ can be computed in $O(M)$ time. Using the linear-time prover for both the NTT and the sparse matrix evaluation protocol, the total prover runtime is $O(N + M)$. The verifier runs in $O(\log^2 N + \log M)$ time.

## C.5 Proof of Theorem 4.3

The accumulator update involves only Hadamard products, NTT, and gadget decomposition, along with simple linear operations. Hence, Construction 7 defines a public-coin PIOP for the accumulator update relation $\mathcal{R}_{\mathsf{acc}}$, reducing to arguments for the relation corresponding to these operations. Perfect completeness follows from the correctness of the computation. By applying a union bound to the soundness error introduced by each argument, we conclude that the overall soundness error is $O(\ell N)/|\mathbb{EF}|$.

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

1. *$\mathcal{P}$ unties $(\mathsf{mult}, \mathsf{Imult}, \mathsf{bits}, \mathsf{Nbits}, \mathsf{prod}) := \mathsf{trace}$.*
2. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) =$*
   - *$(\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{d}, \mathsf{acc}.\boldsymbol{a}, \mathsf{mult}.\boldsymbol{a}))$*
   - *$(\perp, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{d}, \mathsf{acc}.\boldsymbol{b}, \mathsf{mult}.\boldsymbol{b}))$*
   - *$(\perp, (\mathbb{F}, \mathbb{EF}, N, 2\ell), (\mathsf{Nbits}[0..2\ell], \mathsf{Z}.\vec{\boldsymbol{a}}, \mathsf{prod}.\boldsymbol{a}))$*
   - *$(\perp, (\mathbb{F}, \mathbb{EF}, N, 2\ell), (\mathsf{Nbits}[0..2\ell], \mathsf{Z}.\vec{\boldsymbol{b}}, \mathsf{prod}.\boldsymbol{b}))$*

   *In the subclaim, $\mathcal{V}$ queries at a random point, obtaining the evaluations at point $u \xleftarrow{\$} \mathbb{EF}^{\log N}$: $e^{\mathsf{mult}.\boldsymbol{a}} = \mathsf{mult}.\boldsymbol{a}(u)$, $e^{\mathsf{mult}.\boldsymbol{b}} = \mathsf{mult}.\boldsymbol{b}(u)$, $e_i = \mathsf{Nbits}[i](u)$ for each $i \in [2\ell]$. These queried values, along with the random point, are then used to instantiate the subsequent NTT protocol.*

   *// Note that the query $e^{\boldsymbol{d}} = \boldsymbol{d}(u)$ can also be used to instantiate the batched NTT for monomials (Mono-NTT protocol) in Construction 9 for $\mathcal{R}_{\mathsf{NAND}}$.*
3. *$\mathcal{P}$ and $\mathcal{V}$ run the NTT protocol using the batching technique with*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, u, e^{\mathsf{mult}.\boldsymbol{a}}), (\mathsf{Imult}.\boldsymbol{a}))$*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, u, e^{\mathsf{mult}.\boldsymbol{b}}), (\mathsf{Imult}.\boldsymbol{b}))$*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, u, e_i), (\mathsf{bits}[i]))$ for $i \in [2\ell]$*
4. *$\mathcal{P}$ and $\mathcal{V}$ run the gadget decomposition protocol with*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell), (\mathsf{Imult}.\boldsymbol{a}, \mathsf{bits}[0..\ell]))$*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell), (\mathsf{Imult}.\boldsymbol{b}, \mathsf{bits}[\ell..2\ell]))$*
5. *$\mathcal{V}$ samples a random point and check $e^{\mathsf{acc}'.\boldsymbol{a}} = e^{\mathsf{acc}.\boldsymbol{a}} + e^{\mathsf{prod}.\boldsymbol{a}}$, $e^{\mathsf{acc}'.\boldsymbol{b}} = e^{\mathsf{acc}.\boldsymbol{b}} + e^{\mathsf{prod}.\boldsymbol{b}}$ with corresponding queries.*

Since each argument has a linear-time prover, the overall prover runtime is $O(\ell N)$. Notably, the $2\ell + 2$ NTT instances are verified using the batching technique described in Section 3.2, allowing the verifier to run in $O(\log^2 N + \ell)$ time, where $\ell$ comes from the query complexity. The total communication cost is also $O(\log^2 N + \ell)$.

## C.6 Proof of Theorem 4.5

To prove Theorem 4.5, we first provide Construction 8, which defines a public-coin PIOP for the modulus switching relation $\mathcal{R}_{\mathsf{switch}}$ with perfect completeness and a soundness error of $O(\ell N)/|\mathbb{EF}|$. Perfect completeness follows from Theorem 4.4, which establishes a dichotomy where each case reduces to an equality test. These two equality tests are combined in Step 6 via a sum of two Hadamard products with the indicator $\mathsf{id}_{\mathsf{a}}$. This indicator, defined in Step 1, ensuring that $\mathsf{id}_{\mathsf{a}}(x) = 1$ if and only if $\boldsymbol{a}(x) = k$. For any $x$ where $\mathsf{id}_{\mathsf{a}}(x) = 1$, we have $\boldsymbol{a}(x) = k$ and $\boldsymbol{b}(x) = 0$, which be checked using an equality test with a random linear combination in Step 5, introducing only a negligible soundness error. Otherwise, it reduces to a range check in Step 3, ensuring that $\boldsymbol{c}(x) \in [0, 2k)$, where $\boldsymbol{c}(x) = \boldsymbol{a}(x) - (2\boldsymbol{b}'(x) - 1) \cdot k - 1)$ for any $x$ where $\mathsf{id}_{\mathsf{a}}(x) = 0$. Since $2k = \frac{Q-1}{q}$ is as large as the field when $q$ is relatively small, the range check argument basically reduces to a gadget decomposition argument as described in Section 3.3. This introduces the dominate soundness error of $O(\ell N)$ in the this PIOP. The remaining steps check the relation of $\boldsymbol{b}' \equiv \boldsymbol{b} \mod q$ with a soundness error at most $O(\log N)/|\mathbb{EF}|$. Hence, we conclude that the overall soundness error is $O(\ell N)/|\mathbb{EF}|$

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

1. *Let $k = \frac{Q-1}{2q}$ and $\ell = \lceil \log_B (2k) \rceil$.*
2. *$\mathcal{P}$ computes $\mathsf{id}_{\mathsf{a}}, \mathsf{id}_{\mathsf{b}}, \boldsymbol{b}'$ and $\boldsymbol{c}$ for each $x \in \{0,1\}^{\log N}$:*
   - *$\mathsf{id}_{\mathsf{a}}(x) = \mathbb{1}(\boldsymbol{a}(x) = k)$*
   - *$\mathsf{id}_{\mathsf{b}}(x) = \mathbb{1}(\boldsymbol{b}(x) = 0)$*
   - *$\boldsymbol{b}'(x) = \mathsf{id}_{\mathsf{b}}(x) \cdot q + \boldsymbol{b}(x)$*
   - *$\boldsymbol{c}(x) = (1 - \mathsf{id}_{\mathsf{a}}(x)) \cdot (\boldsymbol{a}(x) - (2\boldsymbol{b}'(x) - 1) \cdot k - 1)$*

   *and sends $\mathsf{id}_{\mathsf{a}}, \mathsf{id}_{\mathsf{b}}$ and $\boldsymbol{c}$ to the verifier $\mathcal{V}$, and $\boldsymbol{b}'$ can be queried via $\boldsymbol{b}$ and $\mathsf{id}_{\mathsf{b}}$.*
3. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1)(\{\mathsf{id}_{\mathsf{a}}\}, \{1 - \mathsf{id}_{\mathsf{a}}\}, \boldsymbol{0}))$*
   - *$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 1)(\{\mathsf{id}_{\mathsf{b}}\}, \{1 - \mathsf{id}_{\mathsf{b}}\}, \boldsymbol{0}))$*
4. *$\mathcal{P}$ and $\mathcal{V}$ run the range check protocol on $[0, 2k)$ with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, B, \ell, 2k), \boldsymbol{c})$*
5. *$\mathcal{V}$ samples $\lambda_1 \xleftarrow{\$} \mathbb{EF}$ and $\lambda_2 \xleftarrow{\$} \mathbb{EF}$ and sends them to $\mathcal{P}$.*
6. *Let $\mathsf{id}_{\mathsf{point}} = \lambda_1 \cdot (\boldsymbol{a} - k) + \lambda_2 \cdot \boldsymbol{b}$ be an oracle that can be queried via $\boldsymbol{a}$ and $\boldsymbol{b}$.*
7. *$\mathcal{P}$ and $\mathcal{V}$ run Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, \mathbb{EF}, N, 2), (\{\mathsf{id}_{\mathsf{a}}, 1 - \mathsf{id}_{\mathsf{a}}\}, \{\mathsf{id}_{\mathsf{point}}, (\boldsymbol{a} - (2\boldsymbol{b}' - 1) \cdot k - 1 - \boldsymbol{c})\}, \boldsymbol{0}))$.*

Step 1 and 5 can be computed in linear time. The dominant computational cost arises in Step 3, where the range check argument reduces to a gadget decomposition. Since $\ell$ denotes the length of the decomposed gadgets, the prover's runtime is $O(\ell N)$. The verifier's runtime is $O(\log N + \ell)$ where $\ell$ comes from the query complexity.

## C.7 Proof of Theorem 4.6

As illustrated in Fig. 1, the FHE NAND mainly consists of four steps: LWE addition, NTT operations on monomials, accumulator updates and modulus switching. To ensure the correctness of each essential step in Algorithm 3, the PIOP for the FHE NAND relation $\mathcal{R}_{\mathsf{NAND}}$, defined in Construction 9, also incorporates several Hadamard subprotocols to verify the remaining lightweight operations.

Notably, the $n$ accumulator updates are both the most computationally intensive and the dominant contributor to the soundness error. By applying a union bound over the soundness error introduced by each individual argument, we conclude that the overall soundness error is $O(\ell N n)/|\mathbb{EF}|$. The detailed complexity and soundness analysis is given in Section 4.5.

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

1. *$\mathcal{P}$ unties $(\boldsymbol{a}, \boldsymbol{b}, \{\boldsymbol{d}_i\}_{i=0}^{n}, \{\mathsf{acc}[i], \mathsf{tc}[i]\}_{i=0}^{n-1}) := \mathsf{traces}$*
2. *$\mathcal{P}$ and $\mathcal{V}$ run the LWE addition protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (q, n+1, \mathsf{ct}_0, \mathsf{ct}_1), \boldsymbol{a})$.*
3. *Let $\mathsf{aux}_{\mathsf{b}} = (0, \ldots, 0, 1) \in \mathbb{Z}_2^{n+1}$.*
4. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, N, 2), (\{q - \boldsymbol{a}, \boldsymbol{a}\}, \{1 - \mathsf{aux}_{\mathsf{b}}, \mathsf{aux}_{\mathsf{b}}\}, \boldsymbol{b}))$.*
5. *$\mathcal{P}$ computes $\boldsymbol{k} \in \{0,1\}^{n+1}$ and $\boldsymbol{r} \in \mathbb{Z}_N^{n+1}$ such that $\boldsymbol{b} \cdot \frac{2N}{q} = \boldsymbol{k} \cdot q + \boldsymbol{r}$ and sends $\boldsymbol{k}, \boldsymbol{r}$ to $\mathcal{V}$ as oracles.*
6. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, N, 1), (\{\boldsymbol{k}\}, \{1 - \boldsymbol{k}\}, 0))$.*
7. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, N, 1), (\boldsymbol{v}, \boldsymbol{d}_n, \mathsf{acc}[0].\boldsymbol{b}))$*
8. *For $i \in [n]$: $\mathcal{P}$ and $\mathcal{V}$ run the accumulate update protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\mathsf{Z}, (\mathbb{F}, \mathbb{EF}, F^{\mathsf{R}}, N, B, \ell), (\mathsf{acc}[i], \boldsymbol{d}_i - 1, \mathsf{acc}[i+1], \mathsf{tc}[i]))$ // Assuming the underlying sum-check protocols in Step 7 and 8 are executed in parallel, the returned subclaims consists of evaluation queries of the form $\tilde{\boldsymbol{d}}_i(u) \stackrel{?}{=} e_i$ for each $i \in [n+1]$, where $u \in \mathbb{EF}^{\log N}$ is a shared random point.*
9. *Let $\boldsymbol{e}(x) = \sum_{y \in \{0,1\}^{\log(n+1)}} \tilde{eq}(x, y) \cdot e_i$ and sends it to $\mathcal{V}$. // The simple equality test between $\boldsymbol{e}$ and $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_n$ is omitted.*
10. *$\mathcal{P}$ and $\mathcal{V}$ run the Mono-NTT protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, N, n+1, u), (1 - 2\boldsymbol{k}, \boldsymbol{r}, \boldsymbol{e})))$*
11. *$\mathcal{P}$ and $\mathcal{V}$ run the modulus switching protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, Q, q, N, B, \mathsf{ct}.\boldsymbol{a}), \mathsf{acc}[n].\boldsymbol{a})$*
12. *Let $\mathsf{aux}_{\mathsf{b}'} = (1, 0, \ldots, 0)$ and $\boldsymbol{b}' = (\mathsf{ct}.b, 0, \ldots, 0)$.*
13. *$\mathcal{P}$ computes $\boldsymbol{b}_{\mathsf{zero}} = (\mathsf{acc}[n].b_0 + Q/8, 0, \ldots, 0)$ and sends it to $\mathcal{V}$.*
14. *$\mathcal{P}$ and $\mathcal{V}$ run the Hadamard protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, N, 1), (\{\mathsf{acc}[n].\boldsymbol{b} + Q/8\}, \{\mathsf{aux}_{\mathsf{b}'}\}, \boldsymbol{b}_{\mathsf{zero}}))$*
15. *$\mathcal{P}$ and $\mathcal{V}$ run the modulus switching protocol with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \mathbb{EF}, Q, q, N, B, \boldsymbol{b}'), \boldsymbol{b}_{\mathsf{zero}})$*

# D MODULUS SWITCHING

## D.1 Assumption of $2q \mid (Q-1)$

This assumption is easily satisfied under standard FHEW/TFHE parameters. Specifically, $2N \mid (Q-1)$ due to requirements from the NTT operation, and $q \mid 2N$ as dictated by the lift operation that maps each ring element in the LWE ciphertext to a monomial in the RLWE domain (see Section 4.2 for details). From a practical standpoint, $q$ is typically chosen as a power of two, and $Q$ is selected in the form $Q = 2^{\ell} - i \cdot (2N) + 1$ for some integer $i$, where $\ell$ denotes the bit-length of the prime $Q$. This ensures that $2q \mid (Q-1)$, enabling an efficient PIOP for modulus switching in our work..

## D.2 Proof of Theorem 4.5

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 $Q - 1 = 2kq$.

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 $\lfloor \frac{a \cdot q}{Q} \rceil = 0 = b \bmod q$.

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. □

# E NTT/INTT

## E.1 Idea of Delegation Protocol

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 [48] 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 $A_{F^{\mathsf{R}}}^{(i-1)}(\cdot)$:

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

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

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

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

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

## E.2 Full Derivation of Eq. (5)

$$\tilde{F}^{\mathsf{R}}(u,x) = \sum_{y\in\{0,1\}^{\log N}}\tilde{eq}(u,y)\,\tilde{F}^{\mathsf{R}}(y,x) \tag{10}$$

$$= \sum_{y\in\{0,1\}^{\log N}}\tilde{eq}(u,y)\,\omega^{(2\mathcal{Y}^{\mathsf{R}}+1)\cdot\mathcal{X}} \tag{11}$$

$$= \omega^{\mathcal{X}}\cdot\sum_{y\in\{0,1\}^{\log N}}\tilde{eq}(u,y)\,\omega^{\mathcal{X}\cdot 2\mathcal{Y}^{\mathsf{R}}} \tag{12}$$

$$= \omega^{\mathcal{X}}\cdot\sum_{y\in\{0,1\}^{\log N}}\tilde{eq}(u,y)\,\omega^{\mathcal{X}\cdot(\sum_{i=0}^{\log N-1}2^{\log N-i}\cdot y_i)} \tag{13}$$

$$= \omega^{\mathcal{X}}\cdot\sum_{y\in\{0,1\}^{\log N}}\prod_{j=0}^{\log N-1}((1-u_j)(1-y_j)+u_jy_j) \cdot\prod_{i=0}^{\log N-1}\left(\omega^{2^{\log N-i}}\right)^{\mathcal{X}\cdot y_i} \tag{14}$$

$$= \omega^{\mathcal{X}}\cdot\sum_{y\in\{0,1\}^{\log N}}\prod_{i=0}^{\log N-1}((1-u_i)(1-y_i)+u_iy_i)\cdot\omega^{\mathcal{X}\cdot y_i}_{2^{i+1}} \tag{15}$$

$$= \omega^{\mathcal{X}}\cdot\prod_{i=0}^{\log N-1}\sum_{y_i\in\{0,1\}}((1-u_i)(1-y_i)+u_iy_i)\cdot\omega^{\mathcal{X}\cdot y_i}_{2^{i+1}} \tag{16}$$

$$= \omega^{\mathcal{X}}\cdot\prod_{i=0}^{\log N-1}(1-u_i+u_i\cdot\omega^{\mathcal{X}}_{2^{i+1}}) \tag{17}$$

$$= \omega^{\sum_{j=0}^{\log N-1}2^j\cdot x_j}\cdot\prod_{i=0}^{\log N-1}(1-u_i+u_i\cdot\omega^{\mathcal{X}}_{2^{i+1}}) \tag{18}$$

$$= \prod_{i=0}^{\log N-1}(1-u_i+u_i\cdot\omega^{\mathcal{X}}_{2^{i+1}})\cdot\omega^{2^i\cdot x_i} \tag{19}$$

# F DEFINITION OF HASTEBOOTS

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

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

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

# G EVALUATION

## G.1 Asymptotic Complexity

We analyze the asymptotic complexity for all atomic and FHE operations in Table 4. In our implementations for a single NAND operation using the parameter settings described above, we compute the following $L_1=5$, $L_2=3$, $M_1\approx 4L_1n+2n=22528$, $M_2\approx 2L_1n=10240$, $M_3=n+1=1025$. A single NAND operation involves approximately $n(2L1+2)=12288$ NTT instances. The complexity of the Brakedown PCS, as described in [41], leads to $O(N)$ proving time, and $O(\sqrt{N})$ verification time and proof size, where $N$ denotes the size of the polynomial. In comparison, the BaseFold PCS [68] offers $O(N\log N)$ proving time and $O(\log N)$ verification time and proof size.

## G.2 Parameter Comparison to [61]

The parameters in our work are not exactly the same as those in [61], but they are quite similar, as many parameters in FHE need to be tweaked and optimized to ensure both correctness and security. In [61], 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.