

PRISM: Efficient zkSNARKs for RNS-Based Homomorphic Encryption

Zhelei Zhou
Ant Group
Hangzhou, China
zhouzhelei.zzl@antgroup.com

Yun Li
Ant Group
Beijing, China
liyun24@antgroup.com

Zhaomin Yang
Ant Group
Hangzhou, China
yangzhaomin.yzm@antgroup.com

Cheng Hong*
Ant Group
Beijing, China
vince.hc@antgroup.com

Tao Wei
Ant Group
Hangzhou, China
lenx.wei@antgroup.com

Abstract

Homomorphic Encryption (HE) enables computations on encrypted data without decryption, but it does not guarantee the integrity or correctness of the performed operations. To address this limitation, verifiable HE (vHE) has been proposed. However, achieving efficient vHE for RNS-based HE schemes (e.g., BFV/BGV/CKKS) remains challenging: While the Residue Number System (RNS) boosts performance of HE via multi-modulus ciphertext representations, it significantly complicates the cross-field consistency checks in vHE. We observe that existing vHEs for RNS-based HE suffer from at least one of the following limitations: they do not readily extend to the zero-knowledge setting (Atapoor et al., CiC 2024), incur linear proof size & verifier cost (Zhou et al., S&P 2025), or are designed for a modified HE scheme (Cascudo et al., Crypto 2025).

We present PRISM, the first practical zkSNARK for standard RNS-based HEs. Our techniques are threefold: (1) a new cryptographic primitive called *Multiple-Field Polynomial Commitment Scheme* (MF-PCS) that efficiently prove the cross-field modulo relations, which is the key bottleneck in RNS-based HE verification; (2) a novel Polynomial Interactive Oracle Proof (PIOP) for (inverse) number theoretic transforms with $O(N)$ prover time and $O(\log N)$ verifier time in a model with an offline phase; (3) upgrading MF-PCS and PIOPs to achieve zero-knowledge with small overhead via Vector Oblivious Linear Evaluation (VOLE) correlations. We fully implemented PRISM and evaluated it against state-of-the-art schemes. Compared to Zhou et al. which has the fastest prover time, PRISM has $3.5\times$ slower prover time, but up to $7.8\times$ faster verifier time and $7.8\times$ smaller proof size. Compared to Atapoor et al. which has the smallest proof size, PRISM has $5.5\times$ larger proof size, but roughly $10.1\times$ faster prover time and $2.6\times$ faster verifier time.

CCS Concepts

• Security and privacy → Cryptography.

*Cheng Hong is the corresponding author.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

CCS '26, The Hague, Netherlands

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11

<https://doi.org/10.1145/3830454.3846611>

Keywords

Homomorphic Encryption, Zero-Knowledge Proof, SNARK

ACM Reference Format:

Zhelei Zhou, Yun Li, Zhaomin Yang, Cheng Hong, and Tao Wei. 2026. PRISM: Efficient zkSNARKs for RNS-Based Homomorphic Encryption. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 24 pages. <https://doi.org/10.1145/3830454.3846611>

1 Introduction

Homomorphic Encryption (HE) [33] enables computation to be performed directly on encrypted data without decryption, making it a foundational tool for privacy-preserving applications. Among various HE schemes, those based on Ring-LWE (RLWE) (e.g., BGV [12], BFV [27] and CKKS [19]) have attracted significant attention due to their support for Single Instruction Multiple Data (SIMD) operations [51] and their practical efficiency.

However, HE by itself only guarantees *privacy* of the encrypted data, and it does not ensure *integrity* of the computation. Consider the outsourced computation setting, where a resource-constrained client delegates expensive HE evaluation to an untrusted cloud server. A malicious server could deviate from the prescribed computation and return an incorrect ciphertext, which the client would decrypt without detecting the manipulation. To address this, a line of *verifiable HE* (vHE) works [4, 10, 14, 29, 31, 53, 62] have been proposed. These works use Verifiable Computation (VC) or Zero-Knowledge Proofs (ZKP) [34] to let the server prove the homomorphic evaluation has been correctly performed, so that the client can verify the result efficiently.

The Residue Number System (RNS) in HE. Current lattice-based HE schemes typically operate over a very large ciphertext modulus, making direct arithmetic operations computationally expensive. The RNS representation addresses this challenge by representing large ciphertext coefficients modulo a large integer ($Q_L = \prod_{i \in [L]} q_i$) as residues modulo smaller primes q_i . This decomposition transforms expensive big-integer arithmetic into efficient parallel operations over small moduli, significantly improving both performance and scalability. RNS has become the implementation backbone for modern HE schemes in practice, including BFV, BGV, and CKKS, and is widely adopted in popular HE libraries such as Microsoft SEAL [47], HELib [38], OpenFHE [6], and Lattigo [1].

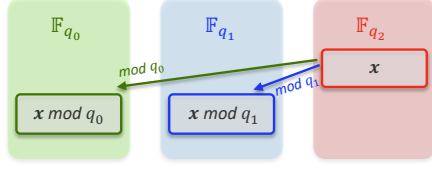


Figure 1: Illustration of the cross-field modulo operation.

The cross-field consistency challenge. A natural proving paradigm for RNS-based HE is the *per-field proving paradigm*: instead of constructing a single expensive proof over the large ring R_Q , one generates independent proofs for each small field $\mathbb{F}_{q_i}$. This paradigm was first proposed by Atapoor et al. [4] and later adopted by ZHE [62]. However, per-field proving becomes problematic for critical HE operations such as modulus switching and key switching, which require *cross-field computations*. Take the CKKS modulus switching from Q_L to $Q_{L-1} = Q_L/q_L$ as an example: Given a ciphertext $ct \in R_{Q_L}^2$, the algorithm computes $\lfloor \frac{ct}{q_L} \rfloor = \frac{ct - (ct \bmod q_L)}{q_L} \bmod Q_{L-1}$, which requires computing $(ct \bmod q_L) \bmod Q_{L-1}$. Since RNS represents ct as L vectors over $\mathbb{F}_{q_1}, \dots, \mathbb{F}_{q_L}$, if the vector over $\mathbb{F}_{q_L}$ is denoted as x , then we have $x = ct \bmod q_L$. Finally, computing $x \bmod Q_{L-1}$ amounts to computing $x \bmod q_i$ for each $i \in [L-1]$, which is the cross-field operation illustrated in Figure 1. We present more details about the cross-field operation in Appendix D.6.

Such cross-field operations bring up the challenge of *cross-field consistency*: the values in one field must be shown to be consistent with the values from a different field. Existing approaches attempt to address this challenge in several ways:

- ABPS [4] exposes these cross-field values as public inputs and let the verifier check the consistency outside the SNARK proof. As a result, ABPS only provides verifiable computation and *does not* readily extend to the ZK setting. This limitation makes it unsuitable for HE applications with server-side private inputs, such as private set intersection and machine learning inference, where ZK property is required to protect the server’s inputs in addition to vHE.
- ZHE [62] constructs a set of Vector Oblivious Linear Evaluation (VOLE) based ZKPs for RNS-based HE, including the cross-field operations, at the cost of linear verifier time and linear proof size. Moreover, proving the cross-field modulo operations remains the dominant bottleneck in ZHE: experiments show that proving the cross-field modulo operations accounts for over 80% of the total prover time (cf. Appendix E).
- CCCFGS [14] takes a different approach: the authors re-design CKKS so that all ciphertexts stay over a *single fixed ring* even when modulus switching is applied, and construct a SNARK over this ring. This design avoids the need to prove cross-field operations entirely. However, the tradeoff is that the resulting CKKS variant is not directly compatible with applications and libraries built on standard CKKS. Moreover, it loses the core efficiency benefits of modulus switching in CKKS, resulting in larger ciphertexts and slower homomorphic operations. We provide a more detailed discussion in Appendix F.

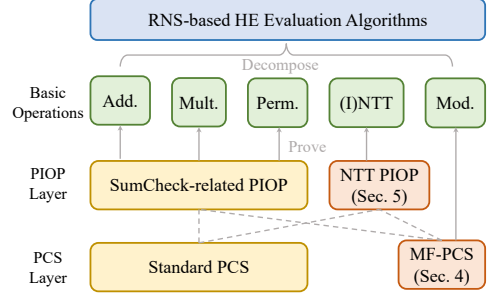


Figure 2: Overview of PRISM.

As a result, existing approaches fail to simultaneously achieve zero-knowledge, succinctness,¹ and compatibility when handling cross-field consistency. Motivated by the above limitations, we ask the following question:

Can we build efficient zkSNARKs for standard RNS-based HE schemes without modifying the underlying HE scheme?

1.1 Our Contributions

We provide an affirmative answer to our research question by presenting an efficient zkSNARK called PRISM (short for “proving RNS-based HE evaluations in a succinct manner”). We follow the modern construction approach: combining Polynomial Interactive Oracle Proofs (PIOPs) and Polynomial Commitment Schemes (PCSs) to obtain SNARKs. More precisely, we follow the per-field proving paradigm and decompose RNS-based HE evaluations into five types of basic operations: additions, multiplications, permutations, Number-Theoretic Transforms (NTT) and its inverse (INTT), and cross-field modulo operations. The first three operations can be handled using existing techniques: combining SumCheck-related PIOPs [16, 43] with standard PCSs. For the latter two operations, we propose a novel PIOP for (I)NTT relations and introduce a new cryptographic primitive called *Multiple-Field PCS* (MF-PCS) to handle cross-field modulo operations. See the overview in Figure 2. Our contributions are summarized as follows.

A new primitive: MF-PCS. We introduce a new cryptographic primitive called *Multiple-Field Polynomial Commitment Scheme* (MF-PCS). Given a set of distinct prime fields $\mathbb{F}_{p_1}, \dots, \mathbb{F}_{p_n}$, MF-PCS allows a prover to commit to an ℓ -variable multilinear polynomial f over $\mathbb{F}_{p_1}$, and later open the extended polynomial $f^{(i)}$ at evaluation points over any of the fields $\mathbb{F}_{p_i}$, where $f^{(i)}$ is a multilinear polynomial over $\mathbb{F}_{p_i}$ such that $f^{(i)}(x) \stackrel{\mathbb{Z}}{=} f(x) \bmod p_i$ for all $x \in \{0, 1\}^\ell$; for $a \in \mathbb{F}_{p_1}, b \in \mathbb{F}_{q_i}$, we write $a \stackrel{\mathbb{Z}}{=} b$ to denote that a and b have the same canonical integer representatives. It is clear that the cross-field consistency checks are captured by MF-PCS. Therefore, MF-PCS can be used to prove the cross-field modulo operations.

We also provide an efficient construction for MF-PCS, which builds upon the well-known Brakedown [35] PCS. Our construction achieves sublinear verifier time and proof size (cf. Section 4) and is concretely efficient (cf. Section 8.1).

¹In this work, we follow Brakedown [35] and define a proof system to be succinct if both the verifier time and proof size are sublinear in the statement size.

Table 1: Comparison between this work and other recent VC/ZKP-based vHE constructions for RNS-based HE.

	ZK	Succinct	Standard HE
ABPS [4]	✗	✓ [†]	✓
ZHE [62]	✓	✗	✓
CCCFGS [14]	✓ [‡]	✓	✗
Ours	✓	✓	✓

[†] In [4], in addition to the SNARK proof, the verifier needs to receive the intermediate cross-field values, which is linear in the witness size, as inputs.

[‡] The original work does not explicitly claim ZK, but we believe their construction can be extended to achieve it.

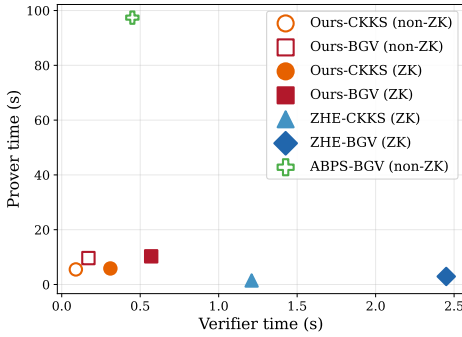


Figure 3: Prover time vs. verifier time for different vHE schemes. Lower-left is better. Solid (resp. hollow) markers indicate schemes with (resp. without) the ZK property.

A novel PIOP for (I)NTT. (I)NTT operations are fundamental operations in RNS-based HE schemes. Prior works have shown that proving (I)NTT requires only $O(N)$ prover time, but they either require linear verifier time [62] or $O(\log^2 N)$ verifier time [42]. We design a new Polynomial Interactive Oracle Proof (PIOP) for (I)NTT that achieves $O(\log N)$ verifier time while maintaining $O(N)$ prover time in a model with an offline phase, making an improvement over prior works [42, 62] (cf. Section 5).

Achieving zero-knowledge via VOLE. We show how to upgrade both MF-PCS and PIOPs to achieve the ZK property using VOLE correlations. Adding ZK via VOLE has two advantages: (i) it introduces only a small overhead on the prover’s computation, since VOLE-based ZKPs are prover-efficient; and (ii) the asymptotic complexity of the *online* verifier time and proof size remain sublinear. The caveats are also two-fold: (i) the resulting zkSNARK is *designated-verifier*, meaning the prover can only prove to a single verifier at a time; (ii) an offline phase for generating random VOLE correlations is required, which is not reusable.

Efficient zkSNARKs for standard RNS-based HE. Combining the above building blocks, we construct efficient zkSNARKs for standard RNS-based HE (cf. Section 7). As shown in Table 1, PRISM is the first to simultaneously achieve zero-knowledge, succinctness, and compatibility with standard RNS-based HE.

Implementation and evaluation. We implement both non-ZK and ZK versions of PRISM for standard CKKS and BGV, and make

comparisons with closely related works. In Figure 3, we provide the prover/verifier time of different vHE works on the simple neural network scenario (cf. Section 8.3); it is clear that PRISM achieves a favorable balance between prover and verifier efficiency.

1.2 Related Work

Here we discuss the related work that is not mentioned previously. **Proving TFHE using SNARK.** A line of work [42, 52, 55] aims to use SNARKs to prove the evaluations of TFHE [22], which is different from the RNS-based HE schemes (BFV/BGV/CKKS) targeted in this work. In [52], Thibault and Walter employ a recursive SNARK to prove the bootstrapping of TFHE, which places a huge burden on the prover’s side. Subsequently, [42] and [55] aim to reduce the prover’s overhead. We note that, in both [42] and [55], the authors prove the modulo operation *within a fixed field* instead of the cross-field modulo operation; therefore, their approaches cannot be extended to prove RNS-based HE evaluations.

Running SNARK inside HE. A different line of work called FHE-SNARK lets the server run the SNARK’s prover algorithm homomorphically on the client’s encrypted input [3, 30, 45, 61]. Then the server returns an encrypted proof to the client, who decrypts it and checks validity using the SNARK’s verification algorithm. While this approach is more efficient in certain settings, it has two limitations. First, as analyzed by [61], this approach is vulnerable to a 1-bit forgery attack, in which a malicious server can extract one bit of the client’s private input by modifying the proof and observing whether the client aborts. Second, as stated in [3], FHE-SNARK cannot support CKKS, which is designed for approximate arithmetic and is widely used in applications such as privacy-preserving machine learning [39, 40]. In contrast, our approach does not suffer from these limitations.

Error-detecting encoding based vHE. Chatel *et al.* proposed new plaintext encoding schemes to construct vHE [15]; due to the error-detecting nature of the proposed encoding schemes, the client can detect the server’s malicious behaviors by checking the encoding of the returned results. However, a recent work [18] pointed out that the security of [15] can be broken by its forgery attacks.

2 Preliminaries

2.1 Notation

Let λ denote the computational security parameter. We denote by $x \leftarrow S$ the uniform sampling of x from a set S . For $a, b \in \mathbb{N}$, we write $[a, b] := \{a, \dots, b\}$; for $a \in \mathbb{N}$, we write $[a] := \{1, \dots, a\}$. We write $\lfloor x \rfloor$ for rounding to the nearest integer, and use bold letters (e.g., $\mathbf{x}$) for vectors. We use $\text{negl}(\cdot)$ for a negligible function, and $\mathcal{X} \stackrel{c}{\approx} \mathcal{Y}$ for computational indistinguishability. The notation $x \stackrel{\mathbb{Z}}{=} y$ denotes equality of the canonical integer representatives of x and y . For an extension field $\mathbb{F}_{p^r}$ of a subfield $\mathbb{F}_p$, we fix a monic irreducible polynomial $f(X)$ of degree r and use $\mathbb{F}_{p^r} \cong \mathbb{F}_p[X]/f(X)$.

We let $B_\mu := \{0, 1\}^\mu$ be the boolean hypercube. For $\mathbf{b} \in B_\mu$, we write $\text{nat}(\mathbf{b}) := \sum_{i \in [\mu]} b_i \cdot 2^{i-1}$. We use $\mathcal{F}_{\mu, p}^{(\leq d)}$ to denote the set of μ -variable polynomials over $\mathbb{F}_p$ whose degree in each variable is at most d . Any polynomial in $\mathcal{F}_{\mu, p}^{(\leq 1)}$ is called multilinear polynomial. Given a function $f : \{0, 1\}^\mu \rightarrow \mathbb{F}_p$, its MultiLinear Extension (MLE) is the unique $\tilde{f} : \mathbb{F}_p^\mu \rightarrow \mathbb{F}_p$ s.t. $\tilde{f}(\mathbf{b}) = f(\mathbf{b})$ for all $\mathbf{b} \in B_\mu$, and

$\tilde{f}(x) = \sum_{b \in B_\mu} f(b) \cdot eq(b, x)$, where $eq(b, x) := \prod_{i \in [\mu]} (b_i \cdot x_i + (1 - b_i) \cdot (1 - x_i))$. MLE of vectors can be defined similarly [58].

2.2 Vector Oblivious Linear Evaluation

Here we introduce VOLE, a.k.a, Information-Theoretic Message Authentication Codes (IT-MACs) [9, 44]. In VOLE, there are two parties: a prover P and a verifier V. Consider a subfield $\mathbb{F}_p$ and its extension field $\mathbb{F}_{p^r}$. For a vector $\mathbf{x} \in \mathbb{F}_p^\ell$ (or $\mathbb{F}_{p^r}^\ell$) held by P, V holds randomly sampled MAC keys $\Delta \in \mathbb{F}_{p^r}$ and $\mathbf{k} \in \mathbb{F}_{p^r}^\ell$, while P holds the corresponding MAC tag $\mathbf{m} \in \mathbb{F}_{p^r}^\ell$ satisfying $\mathbf{m} = \mathbf{k} - \Delta \cdot \mathbf{x}$. Across different instances, Δ can be reused while $\mathbf{k}$ cannot; hence, we refer to Δ as the global MAC key and $\mathbf{k}$ as the local MAC key. We use the notation $\llbracket \mathbf{x} \rrbracket$ to denote such an authenticated vector. The above definitions extend naturally to a single element x , denoted as $\llbracket x \rrbracket$. The VOLE correlations satisfy the following properties:

- *Hiding*. V learns nothing about $\mathbf{x}$ from Δ and $\mathbf{k}$.
- *Binding*. P cannot open $\llbracket \mathbf{x} \rrbracket$ to a different value $\mathbf{x}' \neq \mathbf{x}$ except with probability at most p^{-r} , since P has no knowledge of Δ .
- *Additive Homomorphism*. Given $\llbracket \mathbf{x}_1 \rrbracket, \dots, \llbracket \mathbf{x}_n \rrbracket$ and public coefficients $a_1, \dots, a_n, \mathbf{b}$, both P and V can locally compute $\llbracket \mathbf{y} \rrbracket$ where $\mathbf{y} = \mathbf{b} + \sum_{i=1}^n a_i \cdot \mathbf{x}_i$.

We use the VOLE functionality $\mathcal{F}_{\text{VOLE}}$ [57, 60] to generate VOLE correlations, whose formal description is put in Appendix D.1.

Authentications over different fields. When there are vectors over multiple fields (say $\mathbf{x} \in \mathbb{F}_p^\ell$ and $\mathbf{y} \in \mathbb{F}_q^\ell$), we need to clarify which field is used for each authentication. In this case, we use $\llbracket \mathbf{x} \rrbracket_p$ and $\llbracket \mathbf{y} \rrbracket_q$ to denote such authentications, respectively.

Important procedures. Here we introduce several important procedures that will be used in this work.

- **Auth($\mathbf{x}, \llbracket \mathbf{u} \rrbracket$)**: P sends $\mathbf{d} := \mathbf{x} - \mathbf{u}$ to V, so both parties obtain $\llbracket \mathbf{x} \rrbracket := \llbracket \mathbf{u} \rrbracket + \mathbf{d}$.
- **CheckMult($\llbracket \mathbf{z} \rrbracket, \llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket$)**: This can be used to check if $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$ holds, where $\odot$ denotes the element-wise multiplication. We instantiate this procedure using the QuickSilver protocol [60].
- **CheckZero($\llbracket \mathbf{x} \rrbracket$)**: This can be used to check if $\mathbf{x}$ is a all-zero vector. Let H be a hash function modeled as a random oracle, and let the MAC tag be $\mathbf{m}$ and the local MAC key be $\mathbf{k}$. We instantiate this procedure by letting P compute $\sigma := H(\mathbf{m})$ and send σ to V, who checks if $\sigma = H(\mathbf{k})$ holds.
- **CheckMod($\llbracket \mathbf{x} \rrbracket_p, \{\llbracket \mathbf{y}^{(i)} \rrbracket_{q_i}\}_{i \in [n]}$)**: This can be used to check if $\mathbf{y}^{(i)} = \mathbf{x} \bmod q_i$ holds for each $i \in [n]$. We instantiate this procedure using the ZHE's protocol [62].

2.3 Zero-Knowledge SNARKs

Here we adapt the definition provided in [35]. Consider an NP relation $\mathcal{R} = \{(\mathbf{x}; \mathbf{w})\}$. A SNARK for $\mathcal{R}$ consists of three PPT algorithms (Setup, Prove, Verify): Setup(1^λ) outputs a prover key pk and a verifier key vk ; Prove($\text{pk}, \mathbf{x}, \mathbf{w}$) outputs a proof π ; Verify($\text{vk}, \mathbf{x}, \pi$) outputs a bit b indicating acceptance or rejection. A SNARK satisfies *completeness* (honestly generated proofs are accepted by honest verifiers), *knowledge soundness* (a prover should know a valid witness if she can output an accepting proof), and *succinctness* (verifier time and proof size are sublinear in the statement size). A zkSNARK additionally satisfies *zero-knowledge*: the proof reveals nothing about the witness. The formal definitions are provided in Appendix D.2.

In many SNARKs, Setup is instantiated with Common Reference String (CRS) [32, 36] or Random Oracle (RO) [2]. Recently, several works show that Setup can be instantiated with $\mathcal{F}_{\text{VOLE}}$ [25, 41].

“PCS+PIOP” paradigm. Modern SNARKs often build upon a type of information-theoretic proof system called PIOP [16, 58]. In a PIOP, P and V interact through a series of rounds, where P sends oracles of polynomials to V, and V can query these oracles at arbitrary points. Following [16], we use $\llbracket f \rrbracket$ to denote the oracle of a polynomial f over $\mathbb{F}_p$; when the underlying field needs to be emphasized, we can write $\llbracket f \rrbracket_p$. These polynomial oracles can be instantiated using PCS, which allows P to commit to a polynomial f , and later open $f(x)$ for any evaluation point x . Note that, PIOPs can be made non-interactive by using Fiat-Shamir transformation [5, 28]. The formal definitions of PCS and PIOP are put in Appendix D.3 and D.4, respectively.

2.4 Useful PIOPs

Here we introduce the well-known SumCheck PIOP [43] for $\mathcal{R}_{\text{Sum}}$ and its variants. First of all, we present the definition of $\mathcal{R}_{\text{Sum}}$.

DEFINITION 1. Let $\mathcal{R}_{\text{Sum}}$ be the set of all tuples $(\mathbf{x}; \mathbf{w}) = ((\llbracket f \rrbracket, H, d); f)$ where $f \in \mathcal{F}_{\mu, p}^{(\leq d)}$ and $H = \sum_{b \in B_\mu} f(b)$.

In this work, we focus on small field $\mathbb{F}_p$. To achieve negligible soundness error, we follow [7] and V sample the challenge from a larger extension field $\mathbb{F}_{p^r}$. We describe the protocol as follows:

- For $i \in [\mu]$:
 - If $i = 1$, P computes $f_i(X) := \sum_{b \in B_{\mu-1}} f(X, \mathbf{b})$ over $\mathbb{F}_p$; else, P computes $f_i(X) := \sum_{b \in B_{\mu-i}} f(\alpha_1, \dots, \alpha_{i-1}, X, \mathbf{b})$ over $\mathbb{F}_{p^r}$. After that, P sends $\{f_i(\beta)\}_{\beta \in [0, d] \setminus \{1\}}$ to V.
 - V computes $f_i(1) := H - f_i(0)$, sends a random $\alpha_i \leftarrow \mathbb{F}_{p^r}$ to P, and updates $H := f_i(\alpha_i) \in \mathbb{F}_{p^r}$.
- V accepts if $f(\alpha_1, \dots, \alpha_\mu) = H$.

Variants. Based on SumCheck PIOP, ZeroCheck PIOP for $\mathcal{R}_{\text{Zero}}$ and permutation PIOP for $\mathcal{R}_{\text{Perm}}$ are proposed [16]. In the following, we present the formal descriptions of $\mathcal{R}_{\text{Zero}}$ and $\mathcal{R}_{\text{Perm}}$, the detailed constructions of ZeroCheck/permutation PIOP can be found in [16].

DEFINITION 2. Let $\mathcal{R}_{\text{Zero}}$ be the set of all tuples $(\mathbf{x}; \mathbf{w}) = ((\llbracket f \rrbracket); f)$ where $f \in \mathcal{F}_{\mu, p}^{(\leq d)}$ and $\forall \mathbf{b} \in B_\mu : f(\mathbf{b}) = 0$. Let $\mathcal{R}_{\text{Perm}}$ be the set of all tuples $(\mathbf{i}; \mathbf{x}; \mathbf{w}) = (\sigma; (\llbracket f \rrbracket, \llbracket g \rrbracket); f, g)$ where $\sigma : B_\mu \rightarrow B_\mu$ is a permutation, $f, g \in \mathcal{F}_{\mu, p}^{(\leq d)}$ and $\forall \mathbf{b} \in B_\mu : g(\mathbf{b}) = f(\sigma(\mathbf{b}))$.

In this work, we require each $f \in \mathcal{F}_{\mu, p}^{(\leq d)}$ have the form of $f(X) = g(h_1(X), \dots, h_c(X))$, where $h_i(X) \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ and g is a polynomial of total degree at most d , then there exist PIOPs for $\mathcal{R}_{\text{Sum}}/\mathcal{R}_{\text{Zero}}/\mathcal{R}_{\text{Perm}}$ with $O(N)$ prover time and $O(\log N)$ verifier time [16].

2.5 Linear Error-Correcting Codes

An $[n, k, d]$ -linear error-correcting code C over a finite field $\mathbb{F}$ is a linear subspace of $\mathbb{F}^n$ with dimension k and minimum Hamming distance d . Equivalently, C can be described by an encoding function $\text{Enc} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ that maps a message $\mathbf{u} \in \mathbb{F}^k$ to a codeword $\text{Enc}(\mathbf{u}) \in \mathbb{F}^n$. The rate of the code is defined as $\rho := k/n$. For a code with minimum distance d , the *unique decoding radius* is $t = \lfloor (d-1)/2 \rfloor$. That is, given a received word $\mathbf{r} \in \mathbb{F}^n$ with at most t errors, there

exists a unique codeword within distance t from $\mathbf{r}$. In this work, we focus on codes with the following additional properties:

- The code admits a polynomial-time decoding algorithm Dec.
- The encoding can be expressed as $\text{Enc}(\mathbf{u}) = (\mathbf{u}, \mathbf{p})$ for some parity vector $\mathbf{p} \in \mathbb{F}^{n-k}$, meaning the codeword contains the original message as a prefix.

Extension codes. Given a linear code C over $\mathbb{F}_p$ with generator matrix $G \in \mathbb{F}_p^{k \times n}$, its *extension code* $\hat{C}$ over $\mathbb{F}_{p^r}$ is defined by reusing the same generator matrix G over the extension field [24]. We denote the encoding function of $\hat{C}$ by $\hat{\text{Enc}} : \mathbb{F}_{p^r}^k \rightarrow \mathbb{F}_{p^r}^n$. The extension code $\hat{C}$ preserves the same rate and minimum distance as C .

RS codes. Linear codes can be instantiated using Reed-Solomon (RS) codes. An $[n, k, d]$ -RS code over $\mathbb{F}_p$ is defined as follows: given a message $\mathbf{u} = (u_0, \dots, u_{k-1}) \in \mathbb{F}_p^k$, interpolate a polynomial $P(X) = \sum_{i=0}^{k-1} u_i X^i$ of degree at most $k-1$, and evaluate $P(X)$ at n distinct points $\{\alpha_0, \dots, \alpha_{n-1}\}$ over $\mathbb{F}_p$ to obtain the codeword.

2.6 RLWE-based HE

We focus on RLWE-based HEs, including BGV [12], BFV [27], and CKKS [19]. The detailed HE algorithms are put in Appendix D.5.

Ring structure. A common feature of RLWE-based HEs is that ciphertexts are elements of polynomial rings. Let N be a power of two. We define the polynomial ring $R := \mathbb{Z}[X]/(X^N + 1)$, and for a modulus Q , the quotient ring $R_Q := R/Q \cong \mathbb{Z}_Q[X]/(X^N + 1)$.

RNS representation. In practice, the modulus Q is a product of distinct primes $Q = q_0 \cdot q_1 \cdots q_\ell$, where each q_i is a machine-word-sized prime. The RNS [17] represents a ring element $a \in R_Q$ as $(\ell + 1)$ polynomials $(a_0, \dots, a_\ell)$ with $a_i \in R_{q_i}$ via CRT.

NTT form. Polynomial multiplication is implemented via NTT. For a prime $q_i \equiv 1 \pmod{2N}$ with primitive $2N$ -th root of unity ω_i , we compute the NTT form of $a_i(X) \in R_{q_i}$ as follows: $\text{NTT}(a_i) := (a_i(\omega_i), a_i(\omega_i^3), \dots, a_i(\omega_i^{2N-1})) \in \mathbb{Z}_{q_i}^N$.

3 Technical Overview

3.1 Proving Cross-Field Modulo Operations

During RNS-based HE evaluations, operations such as modulus switching and key switching require proving cross-field modulo operations. For example, given a set of prime-order fields $\{\mathbb{F}_{p_1}, \mathbb{F}_{p_2}, \dots, \mathbb{F}_{p_n}\}$ and a vector $\mathbf{x}^{(1)} = (x_0, x_1, \dots, x_{M-1})$ over the first field $\mathbb{F}_{p_1}$, we need to show $\mathbf{x}^{(i)} \stackrel{\mathbb{Z}}{=} \mathbf{x}^{(1)} \pmod{p_i}$ for each $i \in [2, n]$.

To translate the above instance into the language of ZKPs, we express each $\mathbf{x}^{(i)}$ as a multilinear polynomial $f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}$, where $M = 2^\mu$ is the length of its evaluation vector. The core task becomes: given $\{[f^{(i)}]_{p_i}\}_{i \in [n]}$, prove that $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b}) \pmod{p_i}$ holds for all $\mathbf{b} \in B_\mu$ and $i \in [2, n]$.

First attempt: CMC proofs. Very recently, Wei et al. proposed a primitive called Cross-Modulus Consistent (CMC) proof [56]: given two polynomials over different fields (say, $f \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ and $g \in \mathcal{F}_{\mu, q}^{(\leq 1)}$ where $p > q$), a CMC proof shows that f and g agree as canonical integers on the hypercube, i.e., $\forall \mathbf{b} \in B_\mu, f(\mathbf{b}) \stackrel{\mathbb{Z}}{=} g(\mathbf{b})$. Using this technique, one could prove the cross-field modulo operations as follows: for each $i \in [2, n]$, if $p_1 < p_i$, apply a CMC proof to show $\forall \mathbf{b} \in B_\mu : f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b})$; if $p_1 > p_i$, first compute $g^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}$

where $g^{(i)}(\mathbf{b}) = f^{(1)}(\mathbf{b}) \pmod{p_i}$, then prove the correctness of $g^{(i)}$, and finally apply a CMC proof to show $\forall \mathbf{b} \in B_\mu : f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} g^{(i)}(\mathbf{b})$.

However, this approach suffers from a significant efficiency bottleneck. As noted in [56], each CMC proof requires a range proof, which is computationally expensive. To prove the cross-field modulo operations over n distinct fields, at least $n-1$ range proofs are required, imposing substantial overhead on the prover.

Our approach: embedding modulo relations in PCS. Rather than treating the cross-field modulo operations as constraints to be proved at the PIOP layer, we construct a PCS that can be opened across different fields, thereby embedding the cross-field modulo relations into the PCS itself.

More precisely, we propose a new primitive called *Multiple-Field PCS (MF-PCS)*. Given a set of distinct fields $\{\mathbb{F}_{p_i}\}_{i \in [n]}$ and its extension field $\{\mathbb{F}_{p_i^{r_i}}\}_{i \in [n]}$, MF-PCS allows us to commit to a multilinear polynomial $f^{(1)}$ over $\mathbb{F}_{p_1}$ (which is called the *master field*), and we can open it later at $\mathbf{x}^{(i)} \in \mathbb{F}_{p_i^{r_i}}^\mu$ for each $i \in [n]$.

Note that, when $i \neq 1$, opening $f^{(1)}(\mathbf{x}^{(i)})$ means: (i) derive $f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}$ such that $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b}) \pmod{p_i}$ for all $\mathbf{b} \in B_\mu$; (ii) open $f^{(i)}(\mathbf{x}^{(i)})$. Therefore, the modulo relations are naturally embedded in MF-PCS. We show how to construct MF-PCS efficiently below.

Starting point: Brakedown-style PCS. Our MF-PCS builds upon the Brakedown-style PCS [24, 35] framework, which we review below.

To commit to $f \in \mathcal{F}_{\mu, p}^{(\leq 1)}$, P reshapes the evaluation vector $\mathbf{u} = (f(\mathbf{b}))_{\mathbf{b} \in B_\mu} \in \mathbb{F}_p^M$ into an $m \times m$ matrix U (where $M = m^2$), encodes each row using a linear error-correcting code $\text{Enc} : \mathbb{F}_p^m \rightarrow \mathbb{F}_p^{m'}$ to obtain a codeword matrix C , and commits via a Merkle tree over the columns of C . To open f at $\mathbf{x} \in \mathbb{F}_{p^r}^\mu$, P and V need to run a *testing phase* protocol and an *evaluation phase* protocol. In the testing phase, V sends a random $\mathbf{s} \in \mathbb{F}_{p^r}^m$, P responds with $\mathbf{w} := \mathbf{s}^T \cdot U$, and V asks P to open the j -th column of C (denoted as C^j) to verify $\hat{\text{Enc}}(\mathbf{w})_j = \mathbf{s}^T \cdot C^j$, where j is sampled from a random subset of size $\Theta(\lambda)$. In the evaluation phase, V derives $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{F}_{p^r}^m$ such that $f(\mathbf{x}) = (\mathbf{q}_1 \otimes \mathbf{q}_2)^T \cdot U$, and runs the testing phase protocol with $\mathbf{q}_1$ replacing $\mathbf{s}$; finally, V computes $\mathbf{w}^T \cdot \mathbf{q}_2 = f(\mathbf{x})$.

Our construction blueprint. We extend the Brakedown-style PCS to construct MF-PCS. The key modification is to maintain commitments in each field $\mathbb{F}_{p_i}$ and an additional *auxiliary field* $\mathbb{F}_{p_{n+1}}$ (explained later), then establish consistency across all fields. More precisely, given $f^{(1)} \in \mathcal{F}_{\mu, p_1}^{(\leq 1)}$, P proceeds as follows:

- (1) **Matrix reshaping:** Compute $\mathbf{u} := (f^{(1)}(\mathbf{b}))_{\mathbf{b} \in B_\mu}$ and reshape into $U \in \mathbb{F}_{p_1}^{m \times m}$.
- (2) **Multi-field row encoding:** For each field $i \in [n+1]$:
 - (a) Compute $U^{(i)} := U \pmod{p_i}$ (where for $i=1$, this is simply U itself). Note that the prover can obtain an extended multilinear polynomial $f^{(i)}$ from $U^{(i)}$.
 - (b) Encode each row using a linear error-correcting code $\text{Enc}^{(i)}$ over $\mathbb{F}_{p_i}$, obtaining $C^{(i)} \in \mathbb{F}_{p_i}^{m \times m'}$.
- (3) **Commitment:** Compute Merkle tree commitments to $C := \begin{bmatrix} C^{(1)} \\ \vdots \\ C^{(n+1)} \end{bmatrix} \in \mathbb{Z}^{(n+1)m \times m'}$.
- (4) **Consistency check:** Prove the following:

- (a) For all $i \in [n]$, $\mathbf{b} \in B_\mu$: $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(n+1)}(\mathbf{b}) \bmod p_i$.
- (b) For all $\mathbf{b} \in B_\mu$: $f^{(n+1)}(\mathbf{b}) \in [0, p_1 - 1]$.

(5) **Testing/Evaluation:** Similar to the Brakedown-style PCS.

We now describe how we prove step 4 below, which is inspired by [56]. For step 4a, we introduce an *auxiliary field* $\mathbb{F}_{p_{n+1}}$ to serve as a bridge. Concretely, for each $i \in [n]$ and $j \in [r_i]$, V sends a random vector $\mathbf{s}^{(i,j)} \in \mathbb{F}_{p_i}^m$, and the prover responds with two row vectors: $\mathbf{w}^{(i,j)} := (\mathbf{s}^{(i,j)})^T \cdot \mathbf{U}^{(i)} \in \mathbb{F}_{p_i}^{1 \times m}$ and $\mathbf{v}^{(i,j)} := (\mathbf{s}^{(i,j)})^T \cdot \mathbf{U}^{(n+1)} \in \mathbb{F}_{p_{n+1}}^{1 \times m}$. If we choose $p_{n+1} > \max\{m \cdot p_i^2\}_{i \in [n]}$ to avoid overflow during the computation of $\mathbf{v}^{(i,j)}$, and combine the above with the testing procedure, then $\mathbf{w}^{(i,j)} \stackrel{\mathbb{Z}}{=} \mathbf{v}^{(i,j)} \bmod p_i$ implies with overwhelming probability that $\forall \mathbf{b} \in B_\mu$: $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(n+1)}(\mathbf{b}) \bmod p_i$. The above is still insufficient, as there is no guarantee that $\forall \mathbf{b} \in B_\mu$: $f^{(n+1)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b})$. To address this, we additionally require P to invoke a range proof (e.g., [26]) showing that $\forall \mathbf{b} \in B_\mu$: $f^{(n+1)}(\mathbf{b}) \in [0, p_1 - 1]$. This establishes step 4b.

Optimization: systematic codes. When the underlying linear codes are *systematic* (cf. Section 2.5), our MF-PCS can be further improved. Let $\mathbf{C}^{(i)} = [\mathbf{U}^{(i)} || \mathbf{T}^{(i)}]$ where $\mathbf{T}^{(i)}$ denotes the parity-check matrix, for each $i \in [n+1]$. The key observation is that only $\mathbf{U}^{(1)}$ needs to be committed. When P opens the j -th column $\mathbf{U}^{(1),j}$, V can locally compute $\mathbf{U}^{(i),j} = \mathbf{U}^{(1),j} \bmod p_i$ for each $i \in [2, n+1]$. This reduces the communication by $n \cdot m \cdot |Q_1|$ field elements, where Q_1 denotes the subset of queried columns.

Efficiency. Compared to the CMC proof-based approach which requires $n-1$ range proofs, our approach requires only *one* range proof, which yields substantial efficiency improvements.

3.2 Proving (I)NTT Operations

Starting point: ZHE's approach. ZHE [62] proves NTT relations in the VOLE-ZK setting by exploiting the polynomial identity characterization of NTT. Let $\omega \in \mathbb{F}_p$ be a $2N$ -th root of unity with evaluation domain $H = \{x_j = \omega^{2j+1}\}_{j=0}^{N-1}$. Given authenticated $[\mathbf{u}], [\mathbf{v}]$ with $\mathbf{v} = \text{NTT}(\mathbf{u})$, ZHE's key insight is that the NTT relation holds iff $\mathbf{u}$ and $\mathbf{v}$ (viewed as polynomials) interpolate to the same value at a random challenge s , i.e., $\sum_j s^j \cdot u_j = \sum_j L_j(s) \cdot v_j$, where $L_j(s) := \ell_j^{-1} \cdot (s^N + 1) \cdot (s - x_j)^{-1}$ is the j -th Lagrange basis polynomial over H and $\ell_j := \prod_{k \neq j} (x_j - x_k)$. Both the offline phase (computing $\{\ell_j\}_{j \in [N]}$) and the online phase run in $O(N)$ time. While elegant, this approach has a fundamental limitation: even with an offline phase, the online verifier time is *linear*.

Our approach: achieving sublinear verification. We observe that ZHE's online phase can be reformulated as a PIOP, in this way, the verification time can be reduced to sublinear.

First, we use f, g, e to denote the MLEs of $\mathbf{u}, \mathbf{v}, (\ell_0^{-1}, \dots, \ell_{N-1}^{-1})$, respectively. Then the sum $\sum_j s^j \cdot u_j$ becomes $\sum_{\mathbf{b} \in B_\mu} c_1(\mathbf{b}) \cdot f(\mathbf{b})$, where $c_1(\mathbf{b}) = s^{\text{nat}(\mathbf{b})}$ is a public multilinear polynomial. Second, we use c_2 to denote the MLE of $(L_0(s), L_1(s), \dots, L_{N-1}(s))$, and the correctness of c_2 can be verified by checking $\forall \mathbf{b} \in B_\mu$: $c_2(\mathbf{b}) \cdot (s - x_{\text{nat}(\mathbf{b})}) - (s^N + 1) \cdot e(\mathbf{b}) = 0$, which can be completed via a ZeroCheck PIOP. Finally, we define a polynomial h s.t. $\forall \mathbf{b} \in B_\mu$: $h(\mathbf{b}) = c_1(\mathbf{b}) \cdot f(\mathbf{b}) - c_2(\mathbf{b}) \cdot g(\mathbf{b})$. The NTT relation holds if $\sum_{\mathbf{b} \in B_\mu} h(\mathbf{b}) = 0$, which can be verified by a SumCheck PIOP. The detailed construction is put in Section 5.

3.3 Achieving Zero-Knowledge via VOLE

Making (MF-)PCS zero-knowledge. The key challenge is to hide the committed polynomial while still allowing V to check its algebraic properties. We achieve this by using VOLE correlations.

Specifically, instead of committing to the message matrix $\mathbf{U}^{(1)}$ directly, we let P commit to $\mathbf{D} := \mathbf{U}^{(1)} - \mathbf{A}$, where $[\mathbf{A}]_{p_1}$ is a random VOLE correlation generated in the offline phase. Since $\mathbf{A}$ is uniformly random and unknown to V , $\mathbf{D}$ reveals no information about $\mathbf{U}^{(1)}$. Similarly, P masks the parity-check matrix $\mathbf{T}$ by consuming additional VOLE correlations. When P is asked to open a column (say $\mathbf{U}^{(1),j}$) of the committed matrices, it opens $\mathbf{D}^j$ instead. Both parties then use their VOLE correlations to obtain $[\mathbf{U}^{(1),j}]_{p_1}$, enabling further checks in the authenticated form. Moreover, recall that V sometimes needs to check modulo relations, e.g., $\mathbf{w} = \mathbf{v} \bmod p_1$, where $\mathbf{w} \in \mathbb{F}_{p_1}^\ell$ and $\mathbf{v} \in \mathbb{F}_{p_2}^\ell$. To adapt these checks to the ZK setting, we let P and V invoke CheckMod procedure (cf. Section 2.2).

We note that the standard approach for making Brakedown-style PCS zero-knowledge is to employ randomized linear codes [2, 59]. Compared to our VOLE-based approach, this approach has two limitations: (i) randomized linear codes typically expand the code length, leading to higher costs; (ii) to the best of our knowledge, it is unclear how this approach can be adapted to support checking modulo relations in the zero-knowledge setting.

Making PIOP zero-knowledge. Since all PIOPs in this work build upon SumCheck PIOP, we focus on making SumCheck PIOP zero-knowledge using VOLE correlations. In each round, P sends a univariate polynomial $f_i(X)$ that may leak witness information. Since V only performs *linear* operations on $f_i(X)$, it suffices to let P send $[f_i(X)]$ instead, and V carries out verification using authenticated shares. The detailed construction is in Section 6.2.

4 Multiple-Field PCS

4.1 Definition

Traditional PCS allows us to commit to $f \in \mathcal{F}_{\mu,p}^{(\leq 1)}$, and later open it at an evaluation point over a fixed extension field $\mathbb{F}_{p^r}$. Here we consider a set of distinct fields $\{\mathbb{F}_{p_i}\}_{i \in [n]}$ and their extension fields $\{\mathbb{F}_{p_i^{r_i}}\}_{i \in [n]}$, where n denotes the number of fields. We call $\mathbb{F}_{p_1}$ the *master field*. MF-PCS allows us to commit f over the master field $\mathbb{F}_{p_1}$, and we can open it at an evaluation point over any $\mathbb{F}_{p_i^{r_i}}$, where $i \in [n]$. We put the formal definition of MF-PCS in Figure 4.

4.2 Construction

The intuitions behind our construction can be found in Section 3.1. Here we present the formal description of our MF-PCS construction (without the ZK property) in Figure 5. Note that our construction requires a range proof, which we define as follows. By a (zk)SNARK showing $((\text{com}, \mu, p_1); f^{(n+1)}) \in \mathcal{R}_{\text{Range}}$, we mean a proof of knowledge that: (1) $f^{(n+1)}$ opens the auxiliary-field component of the MF-PCS commitment com ; (2) $\forall \mathbf{b} \in B_\mu$: $0 \leq f^{(n+1)}(\mathbf{b}) < p_1$.

Security analysis. Jumping ahead, we choose to instantiate the linear codes with RS codes. Given that, we prove the security through the following theorem, and the proof is put in Appendix G.1.

THEOREM 1. *If the underlying linear codes are RS codes, then the construction in Figure 5 is an extractable MF-PCS.*

MF-PCS Definition

A MF-PCS for multilinear polynomials is a tuple of five protocols (PCSetup, Commit, Open, VerCom, Eval):

- $(pk, vk) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]})$: It takes the security parameter λ , the number of variables in a polynomial μ , and a set $\{p_i, r_i\}_{i \in [n]}$ which describes the field characteristic as inputs. It outputs the prover key pk and the verifier key vk .
- $(com, h, \{f^{(i)}\}_{i \in [n]}) \leftarrow \text{Commit}(pk, f^{(1)})$: It takes the prover key pk and a polynomial $f^{(1)} \in \mathcal{F}_{\mu, p_1}^{(\leq 1)}$ as inputs, and it outputs a commitment com , the opening hint h and extended polynomials $\{f^{(i)}\}_{i \in [n]}$, where $f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}$.
- $b \leftarrow \text{Open}(vk, com, f^{(1)}, h)$: It takes the verifier key vk , the commitment com , a polynomial $f^{(1)} \in \mathcal{F}_{\mu, p_1}^{(\leq 1)}$ and the opening hint h as inputs, and it outputs a bit b indicating if $f^{(1)}$ is the correct opening of com .
- $b \leftarrow \text{VerCom}(pk, vk, com, \{f^{(i)}\}_{i \in [n]})$: It is a protocol between a PPT prover P and verifier V . Both P and V hold a commitment com . P additionally has n extended polynomials $\{f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}\}_{i \in [n]}$ and the prover key pk , while V additionally holds the verifier key vk . P attempts to convince V that $\forall b \in B_\mu, i \in [2, n] : f^{(i)}(b) \stackrel{\mathbb{Z}}{=} f^{(1)}(b) \bmod p_i$. At the end of the protocol, V outputs a bit b indicating acceptance or rejection.
- $b \leftarrow \text{Eval}(pk, vk, com, \{f^{(i)}, x^{(i)}, y^{(i)}\}_{i \in [n]})$: It is a protocol between a PPT prover P and verifier V ; P and V 's inputs are similar to VerCom, except that both P and V additionally hold $\{x^{(i)} \in \mathbb{F}_{p_i}^\mu, y^{(i)} \in \mathbb{F}_{p_i}^\mu\}_{i \in [n]}$. P attempts to convince V that $f^{(i)}(x^{(i)}) = y^{(i)}$. At the end of the protocol, V outputs a bit b indicating acceptance or rejection.

DEFINITION 3 ((zk) MF-PCS). We say a tuple of five protocols (PCSetup, Commit, Open, VerCom, Eval) is an extractable MF-PCS for multilinear polynomials if the following conditions hold:

- **Completeness:** For any multilinear polynomial over the master field $f^{(1)} \in \mathcal{F}_{\mu, p_1}^{(\leq 1)}$, we have

$$\Pr \left[\begin{array}{l} (pk, vk) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]}); \\ (com, h, \{f^{(i)}\}_{i \in [n]}) \leftarrow \text{Commit}(pk, f^{(1)}) : \end{array} \begin{array}{l} \text{VerCom}(pk, vk, com, \{f^{(i)}\}_{i \in [n]}) = 1 \\ \wedge \text{Eval}(pk, vk, com, \{f^{(i)}, x^{(i)}, y^{(i)}\}_{i \in [n]}) = 1 \\ \wedge \forall i \in [n] : f^{(i)}(x^{(i)}) = y^{(i)} \\ \wedge \forall b \in B_\mu, i \in [2, n] : f^{(i)}(b) \stackrel{\mathbb{Z}}{=} f^{(1)}(b) \bmod p_i \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

- **Binding:** For any PPT adversary A , size parameter $\mu \geq 1$, we have

$$\Pr \left[\begin{array}{l} (pk, vk) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]}); (com, f_0, h_0, f_1, h_1) \leftarrow A(pk); \\ b_0 \leftarrow \text{Open}(vk, com, f_0, h_0); b_1 \leftarrow \text{Open}(vk, com, f_1, h_1) \end{array} : b_0 = b_1 \neq 0 \wedge f_0 \neq f_1 \right] \leq \text{negl}(\lambda).$$

- **Knowledge soundness:** Given $(pk, vk) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]})$, Eval and VerCom together indicate a SNARK for the following NP relation:

$$\mathcal{R}_{KS} = \left\{ \left((com, \{x^{(i)}, y^{(i)}\}_{i \in [n]}); \begin{array}{l} f \in \mathbb{F}_{p_1}[\mu] \wedge \text{Open}(vk, com, f^{(1)}, h) = 1 \\ \wedge \forall i \in [n] : f^{(i)}(x^{(i)}) = y^{(i)}, f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)} \\ \wedge \forall b \in B_\mu, i \in [2, n] : f^{(i)}(b) \stackrel{\mathbb{Z}}{=} f^{(1)}(b) \bmod p_i \end{array} \right) \right\}$$

Moreover, we say a MF-PCS is **zero-knowledge** if (i) com reveals nothing about $f^{(1)}$, i.e., there exists a stateful PPT simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$ such that

$$\left\{ (vk, com) \mid \begin{array}{l} (pk, vk) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]}) \\ (com, h, \{f^{(i)}\}_{i \in [n]}) \leftarrow \text{Commit}(pk, f^{(1)}) \end{array} \right\} \approx \left\{ (vk, com) \mid \begin{array}{l} (pk, vk) \leftarrow \text{Sim}_1(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]}; \rho) \\ com \leftarrow \text{Sim}_2(\rho) \end{array} \right\};$$

(ii) Eval and VerCom together indicate a zkSNARK for $\mathcal{R}_{KS}$.

Figure 4: Definition of MF-PCS

Batching optimization. In vHE, we often need to commit multiple polynomials, so we present the *batched* version of (MF-)PCS as in [42]. Given $B = 2^b$ small polynomials $f_0(y), \dots, f_{B-1}(y)$, each with an evaluation vector of length $M = 2^\mu$, we pack them into a polynomial $F(x, y)$ with $b + \mu$ variables s.t. for every $x \in B_b$, $F(x, y) = f_{\text{nat}(x)}(y)$. By arranging the evaluation vector of F into a $B \times M$ matrix in a column-major order, we can view the evaluation vector of f_i as the i -th row of this matrix. When these small polynomials are opened at the same point r , P first sends $\{f_i(r)\}_{i \in [0, B-1]}$. After receiving another challenge $r' \in \mathbb{F}^b$ from V , P opens $F(r', r)$ to

V , who checks $F(r', r) = \sum_{x \in B_b} eq(x, r') \cdot f_{\text{nat}(x)}(r)$. The approach above can be extended to batched openings at different evaluation points by existing techniques (e.g., Algorithm 4 of HyperPlonk [16]). Furthermore, if the polynomials are opened at a random evaluation point only once, then the testing and the evaluation phase can be merged, as analyzed in [23].

Condition on p_{n+1} . To commit a polynomial of size M using MF-PCS, we require $p_{n+1} > \sqrt{M} \cdot \max_{i \in [n]} \{p_i^2\}$. When batched optimization is considered, i.e., B polynomials of equal size M are packed to be committed at once, we require $p_{n+1} > \sqrt{BM} \cdot \max_{i \in [n]} \{p_i^2\}$.

MF-PCS Construction

Parameter: A set of distinct fields $\{\mathbb{F}_{p_i}\}_{i \in [n]}$ and size parameters M, μ, m . Here M is the length of the evaluation vector committed by MF-PCS; for simplicity, we assume $M = 2^\mu = m^2$.

Primitive: Linear error-correcting systematic $[m', m, d]$ -code $C^{(i)}$ over $\mathbb{F}_{p_i}$ and its extension code $\hat{C}^{(i)}$. The encoding function of $C^{(i)}$ (resp. $\hat{C}^{(i)}$) is denoted as $\text{Enc}^{(i)} : \mathbb{F}_{p_i}^m \rightarrow \mathbb{F}_{p_i}^{m'}$ (resp. $\hat{\text{Enc}}^{(i)} : \mathbb{F}_{p_i}^m \rightarrow \mathbb{F}_{p_i}^{m'}$).

- $(\text{pk}, \text{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu, \{p_i, r_i\}_{i \in [n]})$: Select a prime p_{n+1} such that $p_{n+1} > m \cdot \max_{i \in [n]} \{p_i^2\}$ and $p_{n+1} = \Omega(2^\lambda)$ hold, and set $r_{n+1} = 1$. Check if $p_i^{r_i} = \Omega(2^\lambda)$ for each $i \in [n]$. Select an integer $\gamma = \Theta(\lambda)$. Return $\text{pk} = \text{vk} := (\mu, \gamma, \{p_i, r_i, C^{(i)}\}_{i \in [n+1]})$.
- $(\text{com}, h, \{f^{(i)}\}_{i \in [n]}) \leftarrow \text{Commit}(\text{pk}, f^{(1)})$: given pk and $f^{(1)} \in \mathcal{F}_{\mu, p_1}^{(\leq 1)}$, P works as follows:
 - (1) Let $\mathbf{u} := (f^{(1)}(b))_{b \in B_\mu} \in \mathbb{F}_{p_1}^M$ be the evaluation vector of f over B_μ , then reshape $\mathbf{u}$ into a $m \times m$ matrix $\mathbf{U} \in \mathbb{F}_{p_1}^{m \times m}$. The i -th row of $\mathbf{U}$ is denoted as $\mathbf{U}_i$.
 - (2) For $i \in [n+1]$, compute $\mathbf{U}^{(i)} := \mathbf{U} \bmod p_i$ and obtain the codeword matrix $[\mathbf{U}^{(i)} | \mathbf{T}^{(i)}] \in \mathbb{F}_{p_i}^{m \times m'}$ by setting the j -th row as $\text{Enc}^{(i)}(\mathbf{U}_j^{(i)})$. Notice that, P can obtain the extended polynomial $f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}$ from $\mathbf{U}^{(i)}$.
 - (3) Set a concatenated parity-check matrix $\mathbf{T} := \begin{bmatrix} \mathbf{T}^{(1)} \\ \vdots \\ \mathbf{T}^{(n)} \\ \mathbf{T}^{(n+1)} \end{bmatrix} \in \mathbb{Z}^{(n+1)m \times (m' - m)}$.
 - (4) Compute the Merkle tree commitment c_U to $\mathbf{U}$ and c_T to $\mathbf{T}$.
 - (5) Return $\text{com} := (c_U, c_T)$, the opening hint $h := (\mathbf{U}, \mathbf{T})$ and the extended polynomials $\{f^{(i)}\}_{i \in [n]}$.
- $b \leftarrow \text{Open}(\text{vk}, \text{com}, f^{(1)}, h)$: Parse the opening hint h as two matrices $\mathbf{U}, \mathbf{T}$. Convert $f^{(1)}$ to the matrix $\mathbf{U}' \in \mathbb{F}_{p_1}^{m \times m}$ and obtain the concatenated parity-check matrix $\mathbf{T}'$ as in the Commit protocol. Let $M_1 := \text{dist}(\mathbf{U}, \mathbf{U}')$ and $M_2 := \text{dist}(\mathbf{T}, \mathbf{T}')$, where $\text{dist}(\cdot, \cdot)$ denotes the set of different columns between two matrices. If $|M_1| + |M_2| < d/2$ and c_T, c_U are valid Merkle roots corresponding to $\mathbf{U}$ and $\mathbf{T}$, then output $b := 1$; else, output $b := 0$.
- $b \leftarrow \text{VerCom}(\text{pk}, \text{vk}, \text{com}, \{f^{(i)}\}_{i \in [n]})$: P and V work as follows:
 - (1) For $i \in [n], j \in [r_i]$, V sends random $\mathbf{r}^{(i,j)} \in \mathbb{F}_{p_i}^m$ to P, who returns $\mathbf{t}^{(i,j)} := (\mathbf{r}^{(i,j)})^T \cdot \mathbf{U}^{(i)} \in \mathbb{F}_{p_i}^{1 \times m'}$ and $\mathbf{v}^{(i,j)} := (\mathbf{r}^{(i,j)})^T \cdot \mathbf{U}^{(n+1)} \in \mathbb{F}_{p_{n+1}}^{1 \times m'}$ to V. Then V checks if $\mathbf{t}^{(i,j)} \stackrel{\mathbb{Z}}{=} \mathbf{v}^{(i,j)} \bmod p_i$ holds.
 - (2) For $i \in [n], j \in [r_i]$, V computes $[\mathbf{t}^{(i,j)} | \mathbf{y}^{(i,j)}] := \text{Enc}^{(i)}(\mathbf{t}^{(i,j)}) \in \mathbb{F}_{p_i}^{m'}$ and $[\mathbf{v}^{(i,j)} | \mathbf{z}^{(i,j)}] := \text{Enc}^{(n+1)}(\mathbf{v}^{(i,j)}) \in \mathbb{F}_{p_{n+1}}^{m'}$.
 - (3) V samples two random sets $Q_1 \subset [m]$ and $Q_2 \subset [m' - m]$, where $|Q_1| + |Q_2| = \gamma$. For each k in Q_1 or Q_2 :
 - (a) If $k \in Q_1$, P opens the k -th column of $\mathbf{U}$ (denoted as $\mathbf{U}^k$) by providing the corresponding Merkle path. Then for $i \in [n], j \in [r_i]$, V computes $\mathbf{U}^{(i),k} := \mathbf{U}^k \bmod p_i$, and checks if $\mathbf{t}_k^{(i,j)} = (\mathbf{r}^{(i,j)})^T \cdot \mathbf{U}^{(i),k} \bmod p_i$ and $\mathbf{v}_k^{(i,j)} = (\mathbf{r}^{(i,j)})^T \cdot \mathbf{U}^{(n+1),k} \bmod p_{n+1}$ hold.
 - (b) If $k \in Q_2$, P opens the k -th column of $\mathbf{T}$ by providing the corresponding Merkle path; as a result, V obtains $\mathbf{T}^{(1),k}, \dots, \mathbf{T}^{(n+1),k}$. Then for $i \in [n], j \in [r_i]$, V checks if $\mathbf{y}_k^{(i,j)} = (\mathbf{r}^{(i,j)})^T \cdot \mathbf{T}^{(i),k} \bmod p_i$ and $\mathbf{z}_k^{(i,j)} = (\mathbf{r}^{(i,j)})^T \cdot \mathbf{T}^{(n+1),k} \bmod p_{n+1}$ hold.
 - (4) P invokes a SNARK to convince V that $((\text{com}, \mu, p_1); f^{(n+1)}) \in \mathcal{R}_{\text{Range}}$.
- $b \leftarrow \text{Eval}(\text{pk}, \text{vk}, \text{com}, \{f^{(i)}, \mathbf{r}^{(i)}, \mathbf{y}^{(i)}\}_{i \in [n]})$: This protocol can be divided into two phases:

Testing phase:

 - (1) For $i \in [n+1], j \in [r_i]$, V sends random $\mathbf{s}^{(i,j)} \in \mathbb{F}_{p_i}^m$ to P, who returns $\mathbf{w}^{(i,j)} := (\mathbf{s}^{(i,j)})^T \cdot \mathbf{U}^{(i)} \in \mathbb{F}_{p_i}^{1 \times m'}$.
 - (2) For $i \in [n+1], j \in [r_i]$, V computes $[\mathbf{w}^{(i,j)} | \mathbf{x}^{(i,j)}] := \text{Enc}^{(i)}(\mathbf{w}^{(i,j)}) \in \mathbb{F}_{p_i}^{m'}$.
 - (3) V samples two random sets $Q_1 \subset [m]$ and $Q_2 \subset [m' - m]$, where $|Q_1| + |Q_2| = \gamma$. For each k in Q_1 or Q_2 :
 - (a) If $k \in Q_1$, V asks P to open the k -th column of $\mathbf{U}$ (denoted as $\mathbf{U}^k$). Then for $i \in [n+1], j \in [r_i]$, V computes $\mathbf{U}^{(i),k} := \mathbf{U}^k \bmod p_i$ and checks if $\mathbf{w}_k^{(i,j)} = (\mathbf{s}^{(i,j)})^T \cdot \mathbf{U}^{(i),k} \bmod p_i$ holds.
 - (b) If $k \in Q_2$, V asks P to open the k -th column of $\mathbf{T}$; as a result, V obtains $\mathbf{T}^{(1),k}, \dots, \mathbf{T}^{(n+1),k}$. Then for $i \in [n+1], j \in [r_i]$, V checks if $\mathbf{x}_k^{(i,j)} = (\mathbf{s}^{(i,j)})^T \cdot \mathbf{T}^{(i),k} \bmod p_i$ holds.

Evaluation phase: For $i \in [n]$, let $\mathbf{q}_1^{(i)}, \mathbf{q}_2^{(i)}$ be the vectors such that $f^{(i)}(\mathbf{r}^{(i)}) = (\mathbf{q}_1^{(i)} \otimes \mathbf{q}_2^{(i)})^T \cdot \mathbf{u}^{(i)}$ for $i \in [n]$, where $\mathbf{u}^{(i)}$ denotes the evaluation vector of $f^{(i)}$ over B_μ . The evaluation phase is identical to the testing phase, except that: (1) $\{\mathbf{s}^{(i,j)}\}_{j \in [r_i]} \in \mathbb{F}_{p_i}^m$ is replaced with $\mathbf{q}_1^{(i)} \in \mathbb{F}_{p_i}^m$, and $\text{Enc}^{(i)}$ is replaced with $\hat{\text{Enc}}^{(i)}$; (2) V additionally checks if $\mathbf{y}^{(i)} = (\mathbf{w}^{(i)})^T \cdot \mathbf{q}_2^{(i)} \bmod p_i$ holds. If all checks pass, V outputs 1.

Figure 5: Construction of MF-PCS

Complexity analysis. Let $t_C(m)$ denote the encoding time on a length- m vector over a small field. Let $t_P(M)$, $t_V(M)$, and $p(M)$ denote the prover time, verifier time and proof size of the SNARK for $\mathcal{R}_{\text{Range}}$, respectively. We count a small-field operation as one unit and account explicitly for the auxiliary field. Let $R := \sum_{i=1}^n r_i$. We analyze the complexity of MF-PCS as follows.

Without batching. Recall that $p_{n+1} > \sqrt{M} \cdot \max_{i \in [n]} \{p_i^2\}$, so both computation and communication over the auxiliary field incur an additional $O(\log M)$ factor. The costs of MF-PCS are:

- For Commit: prover time is $O((R + \log M)\sqrt{M}t_C(\sqrt{M}))$ and commitment size is $O(1)$;
- For VerCom and Eval: prover time is $O(R\sqrt{M}\log M + t_P(M))$, verifier time is $O(R\log M t_C(\sqrt{M}) + t_V(M))$, and proof size is $O_\lambda(R\sqrt{M}\log M + p(M))$.

The $O_\lambda(\cdot)$ notation suppresses the factors depending only on λ .

With batching. When B small polynomials of equal size M are packed, the resulting polynomial has size BM . The *amortized costs* per small polynomial are:

- For Commit: prover time is $O((R + \log(BM))\sqrt{M/B}t_C(\sqrt{BM}))$ and the amortized commitment size is $O(1/B)$;
- For VerCom and Eval: the prover time is $O(R\sqrt{M/B}\log(BM) + t_P(BM)/B)$, verifier time is $O((R\log(BM)t_C(\sqrt{BM}) + t_V(BM))/B)$, and proof size is $O_\lambda(R\sqrt{M/B}\log(BM) + p(BM)/B)$.

5 A Novel PIOP for (I)NTT

We start by defining the NTT relation.

DEFINITION 4 (NTT RELATION). *The relation $\mathcal{R}_{\text{NTT}}$ is the set of all tuples $(x; w) = (([f]), [g]); f, g$ where $f, g \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ and $u = \text{NTT}(v)$, where $u_{\text{nat}(b)} = f(b)$, $v_{\text{nat}(b)} = g(b)$ for all $b \in B_\mu$.*

The intuition behind our (I)NTT PIOP construction can be found in Section 3.2. The formal description of our PIOP for $\mathcal{R}_{\text{NTT}}$ is put in Figure 6. The security is proven through Theorem 2, and the proof is put in Appendix G.2.

THEOREM 2. *If $p^* = \Omega(2^\lambda)$, then the construction in Figure 6 is a PIOP for $\mathcal{R}_{\text{NTT}}$.*

Complexity. Since this offline phase is similar to ZHE [62], the offline protocol shares the same time complexity as ZHE (i.e., $O(N)$). Furthermore, this offline phase can be reused for multiple instances. Here, we focus on the online phase. Since P performs $O(N)$ field operations to compute c_1, c_2 and the prover time in ZeroCheck and SumCheck PIOP is also $O(N)$, the total prover time is $O(N)$.

Now we discuss the verifier time complexity. Note that, at the end of the ZeroCheck PIOP, the verifier needs to evaluate $d(r)$, where d is a MLE of $(s - x_0, \dots, s - x_{N-1})$ and $r \in \mathbb{F}_{p^*}^\mu$ denotes the random challenge from the ZeroCheck PIOP. Since d is an MLE, its evaluation at r is given by: $d(r) = \sum_{b \in B_\mu} (s - x_{\text{nat}(b)}) \cdot \tilde{e}q(b, r) = s - \omega \cdot \prod_{i=0}^{\mu-1} (1 + r_i \cdot (\omega^{2^{i+1}} - 1))$. This follows from the fact that $x_{\text{nat}(b)} = \omega \cdot \prod_{i=0}^{\mu-1} (\omega^{2^{i+1}})^{b_i}$, and the MLE of $(\omega^{2^{i+1}})^{b_i}$ at r_i is $(1 - r_i) \cdot 1 + r_i \cdot \omega^{2^{i+1}} = 1 + r_i \cdot (\omega^{2^{i+1}} - 1)$. Therefore, V can compute $d(r)$ in $O(\mu) = O(\log N)$ time. Similarly, V can also compute $c_1(r)$ in $O(\log N)$ time. Given the SumCheck/ZeroCheck PIOP has $O(\log N)$ verifier time, our verifier time is also $O(\log N)$.

NTT PIOP

Parameters: NTT evaluation domain $\{x_j = \omega^{2^{j+1}}\}_{j \in [0, N-1]}$, where $N = 2^\mu$ and $\omega \in \mathbb{F}_p$ is the $2N$ -th root of unity.

Offline phase: P and V compute $\ell_j := \prod_{k \in [0, N-1] \setminus \{j\}} (x_j - x_k)$ for each $j \in [0, N-1]$. P sends $[[e]]$ to V, where $e \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ satisfies $e(b) = \ell_{\text{nat}(b)}^{-1}$ for all $b \in B_\mu$. The correctness of $[[e]]$ can be verified by letting P open e to V.

Online phase:

- (1) V sends a random challenge $s \in \mathbb{F}_{p^*}$ to P.
- (2) Let $c_1, d \in \mathcal{F}_{\mu, p^*}^{(\leq 1)}$ be public multilinear polynomials defined as $\forall b \in B_\mu : c_1(b) = s^{\text{nat}(b)}$, $d(b) = s - x_{\text{nat}(b)}$.
- (3) P sends $[[c_2]]$ to V, where $c_2 \in \mathcal{F}_{\mu, p^*}^{(\leq 1)}$ satisfies $\forall b \in B_\mu : c_2(b) = \frac{s^{N+1}}{d(b)} \cdot e(b)$.
- (4) Let t be a polynomial s.t. $\forall b \in B_\mu : t(b) = c_2(b) \cdot d(b) - (s^N + 1) \cdot e(b)$. P runs a ZeroCheck PIOP to convince V that $([[t]], t) \in \mathcal{R}_{\text{Zero}}$.
- (5) Let h be a polynomial s.t. $\forall b \in B_\mu : h(b) = c_1(b) \cdot f(b) - c_2(b) \cdot g(b)$. P runs a SumCheck PIOP to convince V that $(([[h]], 0); h) \in \mathcal{R}_{\text{Sum}}$.

Figure 6: PIOP construction for $\mathcal{R}_{\text{NTT}}$

Comparison with related work. ZHE [62] provides a VOLE-ZK for (I)NTT operations, but its prover time and verifier time are both $O(N)$. HasteBoots [42] proposes an (I)NTT PIOP with $O(N)$ prover time but $O(\log^2 N)$ verifier time. In contrast, our PIOP achieves $O(\log N)$ verifier time while keeping $O(N)$ prover time in a model with an offline phase. Very recently, a concurrent and independent work [49] also proposed an (I)NTT PIOP with $O(N)$ prover time and $O(\log N)$ verifier time, but without an offline phase. We note that, our (I)NTT PIOP can save N field multiplications during the online phase, when compared to this concurrent work.

Optimization. In vHE, there are multiple (I)NTT instances. For better efficiency, we compress these instances into a single instance via random linear combination as in [42, 55], leveraging the linearity of the (I)NTT operations. We present more details in Appendix D.6.

6 Achieving Zero-Knowledge via VOLE

6.1 Making (MF-)PCS Zero-Knowledge

The intuitions behind our construction can be found in Section 3.3. The formal description of our VOLE-based ZK MF-PCS construction is put in Figure 7. We prove the security through Theorem 3 and the proof is put in Appendix G.3. Note that, our approach can also be applied to standard Brakedown PCS to make it zero-knowledge.

THEOREM 3. *The construction in Figure 7 is a ZK MF-PCS.*

Complexity. Recall that $M = m^2$, $R = \sum_{i \in [n]} r_i$, $t_P(M)$, $t_V(M)$, $p(M)$ denote the prover time, verifier time and proof size of the zk-SNARK for $\mathcal{R}_{\text{Range}}$, respectively. We present the complexity of VOLE-based ZK MF-PCS (without batching) in the following, and the full complexity analysis is put in Appendix H.1; note that, our analysis can be easily extended to the batched setting.

ZK MF-PCS Construction

The ZK MF-PCS construction follows the same structure as Figure 5, with the following modifications.

- **PCSetup:** Same as Figure 5, except that it additionally initializes $\{\mathcal{F}_{\text{VOLE}}^{p_i, r_i}\}_{i \in [n+1]}$ and generates $\llbracket A \in \mathbb{F}_{p_1}^{m \times m} \rrbracket_{p_1}$, $\{\llbracket R^{(i)} \in \mathbb{F}_{p_i}^{m \times (m' - m)} \rrbracket_{p_i}\}_{i \in [n+1]}$, $\{\llbracket \alpha^{(i,j)} \in \mathbb{F}_{p_i}^m \rrbracket_{p_i}, \llbracket \beta^{(i,j)} \in \mathbb{F}_{p_{n+1}}^m \rrbracket_{p_{n+1}}\}_{i \in [n], j \in [r_i]}$, and $\{\llbracket \zeta^{(i,j)} \in \mathbb{F}_{p_i}^m \rrbracket_{p_i}\}_{i \in [2, n+1], j \in [\gamma]}$.
 - **Commit:** P obtains $U^{(1)}, \{T^{(i)}\}, \{f^{(i)}\}$ as in Figure 5, then commits to the masked matrices $D := U^{(1)} - A$ and $E^{(i)} := T^{(i)} - R^{(i)}$ for $i \in [n+1]$ via Merkle trees. The opening hint is $h := (U^{(1)}, \llbracket A \rrbracket_{p_1}, \{T^{(i)}, \llbracket R^{(i)} \rrbracket_{p_i}\}_{i \in [n+1]})$.
 - **Open:** Similar to Figure 5.
 - **VerCom:** P and V work as follows:
 - (1) For $i \in [n], j \in [r_i]$, V sends random $r^{(i,j)} \in \mathbb{F}_{p_i}^m$ to P, who computes $t^{(i,j)} := (r^{(i,j)})^T \cdot U \in \mathbb{F}_{p_i}^{1 \times m}$ and $v^{(i,j)} := (r^{(i,j)})^T \cdot U \in \mathbb{F}_{p_{n+1}}^{1 \times m}$. Then $t^{(i,j)}$ and $v^{(i,j)}$ are authenticated via $\text{Auth}(t^{(i,j)}, \llbracket \alpha^{(i,j)} \rrbracket_{p_i})$ and $\text{Auth}(v^{(i,j)}, \llbracket \beta^{(i,j)} \rrbracket_{p_{n+1}})$. Finally, P and V invoke $\text{CheckMod}(\llbracket t^{(i,j)} \rrbracket_{p_{n+1}}, \llbracket v^{(i,j)} \rrbracket_{p_i})$.
 - (2) For $i \in [n], j \in [r_i]$, V computes $\llbracket t^{(i,j)} \rrbracket_{p_i} || y^{(i,j)} \rrbracket_{p_i} := \text{Enc}^{(i)}(\llbracket t^{(i,j)} \rrbracket_{p_i})$ and $\llbracket v^{(i,j)} \rrbracket_{p_{n+1}} || z^{(i,j)} \rrbracket_{p_{n+1}} := \text{Enc}^{(n+1)}(\llbracket v^{(i,j)} \rrbracket_{p_{n+1}})$.
 - (3) V samples two random sets $Q_1 \subset [m]$ and $Q_2 \subset [m' - m]$, where $|Q_1| + |Q_2| = \gamma$. For each k in Q_1 or Q_2 :
 - (a) If $k \in Q_1$: P opens the k -th column of D by providing the Merkle path, so both parties can obtain $\llbracket U^{(1),k} \rrbracket_{p_1}$. For $i \in [2, n+1]$, P computes $U^{(i),k} = U^{(1),k} \bmod p_i$, runs $\text{Auth}(U^{(i),k}, \llbracket \zeta^{(i,k)} \rrbracket_{p_i})$ and invokes $\text{CheckMod}(\llbracket U^{(1),k} \rrbracket_{p_1}, \llbracket U^{(i),k} \rrbracket_{p_i})$ with V. Then for $i \in [n], j \in [r_i]$, P and V invoke $\text{CheckZero}((r^{(i,j)})^T \cdot \llbracket U^{(i),k} \rrbracket_{p_i} - \llbracket t_k^{(i,j)} \rrbracket_{p_i})$ and $\text{CheckZero}((r^{(i,j)})^T \cdot \llbracket U^{(i),k} \rrbracket_{p_{n+1}} - \llbracket v_k^{(i,j)} \rrbracket_{p_{n+1}})$.
 - (b) If $k \in Q_2$: P opens the k -th column of E by providing the Merkle path, so both parties can obtain $\{\llbracket T^{(i),k} \rrbracket_{p_i}\}_{i \in [n+1]}$. Then for $i \in [n], j \in [r_i]$, P and V invoke $\text{CheckZero}((r^{(i,j)})^T \cdot \llbracket T^{(i),k} \rrbracket_{p_i} - \llbracket y_k^{(i,j)} \rrbracket_{p_i})$ and $\text{CheckZero}((r^{(i,j)})^T \cdot \llbracket T^{(n+1),k} \rrbracket_{p_{n+1}} - \llbracket z_k^{(i,j)} \rrbracket_{p_{n+1}})$.
 - (4) P invokes a zk-SNARK to convince V that $((\text{com}, \mu, p_1); f^{(n+1)}) \in \mathcal{R}_{\text{Range}}$.
 - **Eval:** The evaluation $y^{(i)}$ can be replaced by $\llbracket y^{(i)} \rrbracket_{p_i}$ via Auth procedure, if P does not want to reveal $y^{(i)}$.
- Testing phase:**
- (1) For $i \in [n+1], j \in [r_i]$, V sends random $s^{(i,j)} \in \mathbb{F}_{p_i}^m$ to P, who computes $w^{(i,j)} := (s^{(i,j)})^T \cdot U^{(i)} \in \mathbb{F}_{p_i}^{1 \times m}$. Then $w^{(i,j)}$ can be authenticated with $\text{Auth}(w^{(i,j)}, \llbracket \alpha^{(i,j)} \rrbracket_{p_i})$.
 - (2) For $i \in [n+1], j \in [r_i]$, V computes $\llbracket w^{(i,j)} \rrbracket_{p_i} || x^{(i,j)} \rrbracket_{p_i} := \text{Enc}^{(i)}(\llbracket w^{(i,j)} \rrbracket_{p_i})$.
 - (3) V samples two random sets $Q_1 \subset [m]$ and $Q_2 \subset [m' - m]$, where $|Q_1| + |Q_2| = \gamma$. For each k in Q_1 or Q_2 :
 - (a) If $k \in Q_1$: Both P and V obtains $\{\llbracket U^{(i),k} \rrbracket_{p_i}\}_{i \in [n+1]}$ using the same approach as in VerCom. Then for $i \in [n+1], j \in [r_i]$, invoke $\text{CheckZero}((s^{(i,j)})^T \cdot \llbracket U^{(i),k} \rrbracket_{p_i} - \llbracket w_k^{(i,j)} \rrbracket_{p_i})$.
 - (b) If $k \in Q_2$: Both P and V obtains $\{\llbracket T^{(i),k} \rrbracket_{p_i}\}_{i \in [n+1]}$ using the same approach as in VerCom. Then for $i \in [n+1], j \in [r_i]$, invoke $\text{CheckZero}((s^{(i,j)})^T \cdot \llbracket T^{(i),k} \rrbracket_{p_i} - \llbracket x_k^{(i,j)} \rrbracket_{p_i})$.
- Evaluation phase:** For $i \in [n]$, let $q_1^{(i)}, q_2^{(i)}$ be the vectors such that $f^{(i)}(r^{(i)}) = (q_1^{(i)} \otimes q_2^{(i)})^T \cdot u^{(i)}$. The evaluation phase is identical to the testing phase, except that: (1) $s^{(i,j)}$ is replaced with $q_1^{(i)}$ and $\text{Enc}^{(i)}$ with $\hat{\text{Enc}}^{(i)}$; (2) P and V additionally invoke $\text{CheckZero}((\llbracket w^{(i)} \rrbracket_{p_i})^T \cdot q_2^{(i)} - \llbracket y^{(i)} \rrbracket_{p_i})$. If all checks pass, V outputs 1.

Figure 7: VOLE-based ZK MF-PCS Construction

- **Offline phase:** Both the prover time and the verifier time are $O_\lambda((n + \log M)M + R\sqrt{M} \log M)$;
- **Online phase:** For Commit protocol, the prover time and commitment size remain unchanged from the non-ZK version. For VerCom and Eval protocols, the prover time is $O_\lambda(R\sqrt{M} \log M + t_P(M))$, the verifier time is $O_\lambda(R\sqrt{M}(\log M)^2 + t_V(M))$ and the proof size is $O_\lambda(R\sqrt{M} \log M + p(M))$.

6.2 Making PIOP Zero-Knowledge

The intuitions behind our construction can be found in Section 3.3. Since all PIOPs in this work are built upon the SumCheck PIOP, we focus on making SumCheck PIOP zero-knowledge. We present the protocol description in Figure 8. We prove the security through Theorem 4, and the proof is put in Appendix G.4.

THEOREM 4. *The protocol in Figure 8 is a ZK PIOP.*

Complexity. In our experiments, the polynomial degree d and the extension degree r are set as small constants, i.e., $d = O(1), r = O(1)$. Given these, we present the complexity as follows; the full complexity analysis is put in Appendix H.2.

- **Offline phase:** Both the prover and the verifier's time are $O(\log N)$;
- **Online phase:** The online prover time is $O(N)$, both the online verifier time and the proof size are $O(\log N)$.

7 PRISM for Admissible HE Evaluations

We first review the definition of admissible HE evaluations from [62].

DEFINITION 5. *Let EvalOp be a HE evaluation operation. We say EvalOp is admissible if it consists of some combination of a subset or*

ZK SumCheck

Input: P holds a polynomial $f \in \mathcal{F}_{\mu,p}^{(\leq d)}$ with $N = 2^\mu$. Both parties hold $\llbracket f \rrbracket$ and the authenticated claimed sum $\llbracket H \rrbracket$, where $H = \sum_{b \in B_\mu} f(b)$.

Offline phase: $\mathcal{F}_{\text{VOLE}}^{p,r}$ is initialized, and random VOLE correlations $\llbracket u_1 \in \mathbb{F}_p^d \rrbracket$ and $\{\llbracket u_i \in \mathbb{F}_{p^r}^d \rrbracket\}_{i \in [2,\mu]}$ are generated.

Online phase:

- (1) For $i \in [\mu]$:
 - (a) P computes the round polynomial $f_i(X)$ as in the standard SumCheck PIOP (cf. Section 2.4). Let $f_i := (f_i(\beta))_{\beta \in [0,d] \setminus \{1\}}$ be the evaluation vector on set $[0,d] \setminus \{1\}$. P and V invoke $\text{Auth}(f_i, \llbracket u_i \rrbracket)$.
 - (b) P and V compute $\llbracket f_i(1) \rrbracket := \llbracket H \rrbracket - \llbracket f_i(0) \rrbracket$.
 - (c) V sends a random challenge $\alpha_i \leftarrow \mathbb{F}_{p^r}$ to P.
 - (d) P and V update $\llbracket H \rrbracket := \llbracket f_i(\alpha_i) \rrbracket$, which can be computed locally from $\llbracket f_i(0) \rrbracket, \llbracket f_i(1) \rrbracket, \dots, \llbracket f_i(d) \rrbracket$ and α_i .
- (2) At the end, both parties hold $\llbracket f(\alpha_1, \dots, \alpha_\mu) \rrbracket$. When V has the oracle access of $\llbracket y \rrbracket$ where $y = f(\alpha_1, \dots, \alpha_\mu)$, P and V invoke $\text{CheckZero}(\llbracket f(\alpha_1, \dots, \alpha_\mu) \rrbracket - \llbracket y \rrbracket)$. If all the checks pass, V outputs 1; otherwise, V outputs 0.

Figure 8: VOLE-based ZK SumCheck protocol

all of the following operations: (i) additions, (ii) multiplications, (iii) permutations, (iv) modulo operations, and (v) (I)NTT operations.

Now we present our main theorem below, and the proof is put in Appendix G.5. This theorem shows that our techniques can be used to obtain a zkSNARK for admissible HE evaluations.

THEOREM 5. *For any admissible HE evaluation EvalOp , there exists a zkSNARK for $\mathcal{R}_{\text{EvalOp}} = \{((\text{EvalOp}, o); w) : o = \text{EvalOp}(w)\}$.*

As stated in [62], the HE evaluations (including key/modulus switching) of RNS-HE schemes are admissible. To provide a better understanding, we show the connections between cross-field modulo operations, (I)NTT operations and vHE in Appendix D.6. We leave more discussions about bootstrapping and public verifiability in Appendix I.

8 Implementation and Evaluation

We provide a prototype implementation of PRISM using a single thread in C++ on top of NTL [50], Microsoft SEAL [47] and EMP-toolkit libraries [54]. Our codes can be found in Appendix B. All experiments were conducted on a cloud instance equipped with Intel Xeon Platinum 8575C CPU at 2.80GHz and 128 GB of RAM. Our implementations achieve 128-bit computational security.

Instantiations. We instantiate the RO with the SHA256 hash functions. We instantiate the standard PCS with Brakedown [35]. We instantiate the linear codes with RS codes of code rate 0.5. Although RS-code encoding has $O(M \log M)$ encoding time for input size M , its concrete efficiency is excellent. Within the Brakedown framework, the RS-code instantiation offers competitive running time and smaller proofs than the other linear-code instantiations, as reported in [35]. We instantiate the range proof using the multilinear variant of the LogUp-style lookup PIOP [26] with Brakedown PCS. We also employ the table-decomposition techniques [48] for better efficiency; the details are given in Appendix J.

HE schemes. We implement our protocols for BGV and CKKS evaluation algorithms, as implemented in Microsoft SEAL library [47].

Offline phase. In all of our experiments, we assume the offline materials are generated by a TTP for simplicity. We note that

the offline phase can be instantiated with efficient 2PC protocols (e.g., [11, 46, 57]) as discussed earlier.

Metrics. We mainly consider (1) offline time: the running time of generating offline materials by a TTP; (2) prover/verifier time: the prover’s/verifier’s computation time during the online phase; (3) proof size: the communication cost of the online phase.

8.1 Performance of MF-PCS

We evaluate batched MF-PCS using the HE parameters of the CKKS logistic-regression training scenario in Section 8.3. Specifically, we set the ring dimension $N = 2^{14}$, and hence the length of each small polynomial is $M = N = 2^{14}$. We consider a CKKS modulus chain which contains eight 48-bit RNS primes, and we vary the number of small fields from $n \in [2, 8]$ to cover the different numbers of active RNS primes across the levels of the modulus chain. We set batch size $B = 2^8$ and fix the auxiliary field to be 150-bit, which satisfies $p_{n+1} > \sqrt{BM} \max_{i \in [n]} p_i^2$ for all evaluated configurations.

We run Commit, VerCom and Eval protocols of MF-PCS, then report the amortized results for each small polynomial in Figure 9. **Comparison baseline.** To better illustrate the practical efficiency of our MF-PCS, we consider the following baseline approach: For each of the n fields, it independently commits the B polynomials of size M using batched Brakedown PCS, then proves each of the $n - 1$ cross-field modulo relations using the CMC PIOP as discussed in Section 3.1. Since the dominant cost of CMC PIOP lies in the range proof component, we omit the remaining operations and only report the performance of n Brakedown PCSs and $n - 1$ range proofs². Consequently, the reported costs underestimate the cost of a complete CMC PIOP implementation and therefore favor the baseline. We use the same RS-code rate, polynomial length M , and batch size B for both baseline approach and our MF-PCS. The reported numbers are amortized over B polynomials of size M .

As shown in Figure 9, the prover/verifier time, and proof size of MF-PCS all grow slower than the baseline as the number of fields n

²The CMC PIOP of [56] requires a range proof over the ring. Existing lookup-based range proofs are primarily optimized for fields, and their direct extensions to rings may incur additional overhead [56]. Since we only simulate the CMC-based approach as a baseline, we instead instantiate this component with the field-based range proof used in our MF-PCS construction. This substitution favors the baseline.

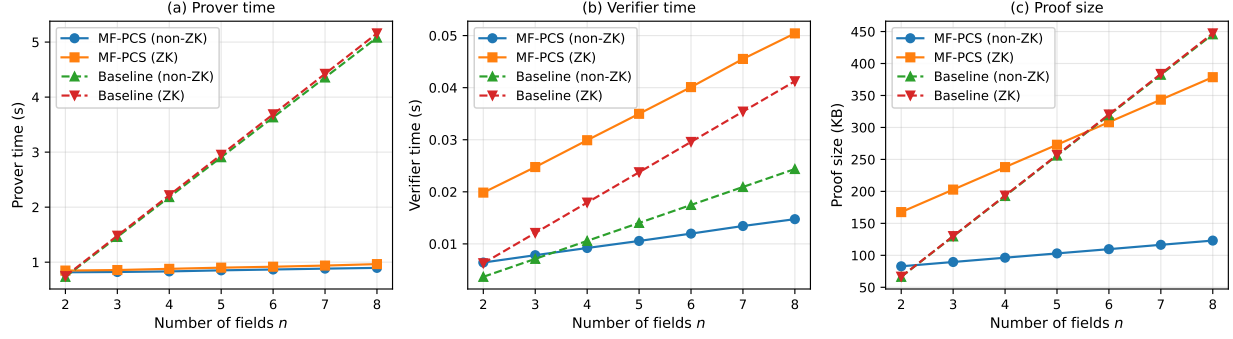


Figure 9: Comparison of MF-PCS with the baseline approach for $B = 2^8$ small polynomials, each with equal size $M = 2^{14}$, varying the number of fields n . All reported numbers are amortized per small polynomial.

increases. This is because the range proof, which is a dominant cost, is invoked only *once* regardless of n in our MF-PCS construction. As a result, our MF-PCS scales gracefully with the number of fields, making it well-suited for RNS-based HE evaluations.

ZK overhead. We analyze the ZK overhead of MF-PCS as follows. *Offline costs.* Our ZK MF-PCS requires an offline phase in which the required VOLE correlations are generated. To show how the offline costs scale with the number of polynomials, we evaluate $B \in \{2^4, 2^6, 2^8, 2^{10}\}$ and $n \in [2, 8]$, while fixing $M = 2^{14}$ and the field parameters. The offline costs are reported in Figure 10. As defined in the metrics above, the reported offline time measures the TTP’s generation time; under the TTP model, the verifier only receives, reads, and stores its shares of the generated correlations.

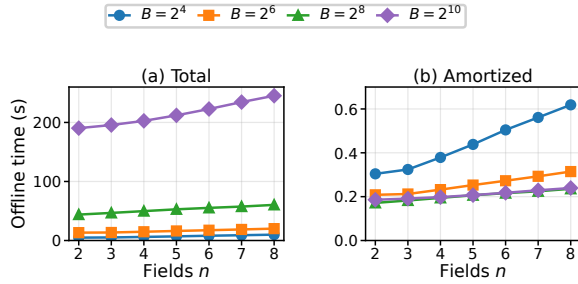


Figure 10: Offline time of ZK MF-PCS for $B \in \{2^4, 2^6, 2^8, 2^{10}\}$ and $M = 2^{14}$: (a) total time; (b) amortized time per polynomial.

Recall that the total amount of offline material to be generated (or, on the verifier side, to be read and stored) is $O_\lambda((n + \log(BM))BM + R\sqrt{BM}\log(BM))$. For fixed M and n , this bound is quasi-linear in B , which is consistent with Figure 10(a), where the total offline time grows as B increases. The amortized offline cost is $O_\lambda((n + \log(BM))M + R\sqrt{M/B}\log(BM))$ per polynomial. Its square-root term decreases with B , while the logarithmic terms grow slowly with B . Figure 10(b) shows diminishing returns as B grows: increasing B from 2^4 to 2^8 substantially reduces the amortized time, whereas the amortized time at $B = 2^{10}$ is slightly higher than that at $B = 2^8$.

We note that, the offline materials are single-use: they are consumed by online proof generation and cannot be reused.

Online costs. The CheckMod invocations account for most of the online ZK overhead. Comparing the ZK and non-ZK MF-PCS results in Figure 9 for $B = 2^8$ and $n \in [2, 8]$, the online prover overhead is only 3.8%–7.6%, since VOLE-ZKPs are prover-efficient. The verifier time grows by $3.1\times$ – $3.4\times$, but remains only 20–50 ms in absolute terms, while the proof size grows by $2.0\times$ – $3.1\times$.

8.2 Performance of (I)NTT PIOP

We compare the concrete performance of our non-ZK (I)NTT PIOP with that of HasteBoots [42]. For fairness, we follow the HasteBoots’ public Rust implementation and re-implement its (I)NTT PIOP in C++, allowing both protocols to be evaluated in the same C++/NTL environment on the same machine. We benchmark a single (I)NTT PIOP instance. We use a 48-bit prime-order field and its degree-three extension field, and vary the size $N \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}\}$. Each experiment is repeated ten times, and we report the average; note that, the reported numbers exclude the costs of PCS.

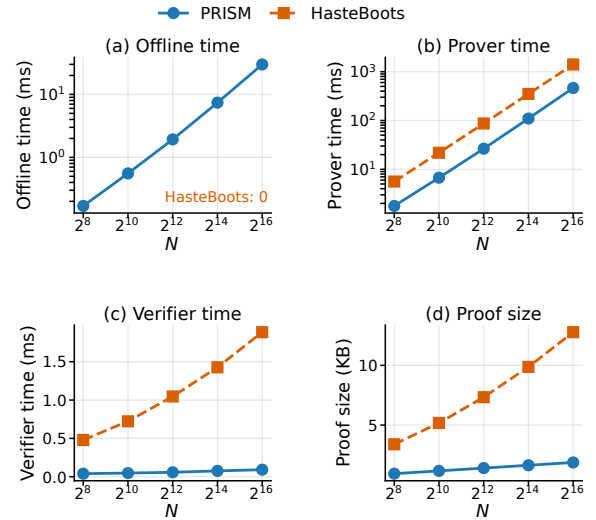


Figure 11: Comparison of (I)NTT PIOPs. The x -axis is logarithmic in all panels. The y -axis is logarithmic in (a) and (b).

As shown in Figure 11, our construction consistently outperforms HasteBoots in all online metrics. Our verifier time is $12.0\times$ – $20.5\times$ faster, while our proof is $3.6\times$ – $6.8\times$ smaller. The growing gaps in verifier time and proof size are consistent with the asymptotic improvement: HasteBoots has $O(\log^2 N)$ verifier time and proof size, whereas both costs are $O(\log N)$ in our PIOP. Although both protocols have $O(N)$ prover time, our prover time is also $3.0\times$ – $3.3\times$ faster, showing a concrete performance advantage. The tradeoff is that our protocol requires an offline phase, whereas HasteBoots has no offline phase. The offline phase in our PIOP is reusable across multiple (I)NTT PIOP instances with the same parameters.

8.3 Performance of PRISM

In the following, we compare PRISM with three related works. We provide more evaluation results in Appendix K.

Comparison with ZHE. We compare the ZK version of PRISM with ZHE by re-running its public implementation on the following scenarios, covering BGV and CKKS.

Ciphertext multiplication (CM). We prove the correctness of a single ciphertext-ciphertext multiplication. For both BGV and CKKS, we set $N = 2^{13}$ and $L = 2$, with 4 primes of 48 bits as the HE modulus. *Matrix-vector multiplication (MVM).* We consider multiplying an encrypted 8×8 matrix with an encrypted length-8 vector, using the diagonal packing technique of [37]. For both BGV and CKKS, we use the same HE parameters as in the CM scenario.

Logistic regression training (LR). We evaluate one iteration of the approximate logistic regression training protocol of [39] using CKKS. We set $N = 2^{14}$ and $L = 6$, with 8 primes of 48 bits as the HE modulus. The number of features is 15 and the number of samples is 512, so that all samples fit in a single CKKS ciphertext.

Comparison approximation (CA). We consider the case where two parties have some private numbers (say, x and y) and want to securely compare whose number is larger. We note that, the ZK property of vHE is necessary in this scenario. We adopt the CKKS-based homomorphic comparison approach of [20]. Given encryptions of x, y , we homomorphically compute an encryption of $z := x - y$, then approximate the comparison result by evaluating a degree-55 polynomial on the encrypted z . The polynomial evaluation has a multiplicative depth of 6. We set $N = 2^{14}$ and $L = 7$, with 9 primes of 48 bits as the HE modulus. Under these HE parameters, 8192 comparisons are evaluated in a batch.

Table 2 summarizes the operation counts in these scenarios. The #MS count includes only standalone modulus switching and excludes that performed inside key switching. In particular, #(I)NTTs is the total number of (I)NTT instances in the HE evaluation; As described in Section 5, we compress them into one by random linear combination and invoke our (I)NTT PIOP once.

The results are shown in Table 3. Across the six scheme–scenario combinations, PRISM’s offline time is $1.1\times$ – $1.8\times$ that of ZHE and its prover is $3.2\times$ – $3.5\times$ slower. In return, PRISM achieves $5.9\times$ – $7.8\times$ faster verification and $4.2\times$ – $7.8\times$ smaller proofs. Moreover, BGV incurs higher proving costs than CKKS because it involves more cross-field modulo relations, each requiring an MF-PCS invocation. **Comparison with ABPS.** We re-ran the public C++ implementation of ABPS to make a comparison. Since ABPS only implements

Table 2: Operation counts in the evaluated scenarios. KS (resp. MS) denotes key (resp. modulus) switching.

Scenario	Scheme	#KS	#MS	#MF-PCS	#(I)NTTs
CM	CKKS	1	0	5	20
	BGV	1	0	9	20
MVM	CKKS	15	0	75	300
	BGV	15	0	135	300
LR	CKKS	21	11	138	724
CA	CKKS	54	54	435	2172

its schemes for BGV and does not provide zero-knowledge, we compare it with our non-ZK version of PRISM for BGV for fairness.

Following ABPS, we consider a two-layer homomorphic circuit that approximates the feedforward evaluation of simple Neural Network (NN) layers. In the first layer, given ciphertexts $\{ct_{0,i}\}_{i \in [5]}$ and plaintexts $\{a_{0,i,j}, b_{0,j}\}_{i \in [5], j \in [3]}$, the circuit computes $ct_{1,j} = (\sum_{i=1}^5 a_{0,i,j} \cdot ct_{0,i} + b_{0,j})^2$ for $j \in [3]$. In the second layer, given $\{ct_{1,i}\}_{i \in [3]}$ and plaintexts $\{a_{1,i}, b_1\}_{i \in [3]}$, the circuit computes $ct_2 = (\sum_{i=1}^3 a_{1,i} \cdot ct_{1,i} + b_1)^2$ as the final output. We use the same BGV parameters as in [4]: $N = 4096$ with the HE modulus being the product of three primes. We report the results in Table 4.

As shown in Table 4, PRISM outperforms ABPS in offline time, prover time, and verifier time, while ABPS has a smaller proof size. We note that PRISM’s offline phase is significantly faster than ABPS’s; this is because PRISM’s non-ZK offline phase only involves the lightweight offline phase of the NTT PIOP, whereas ABPS must generate a lattice-based CRS of 11.6 GB.

Comparison with CCCFGS. We also provide a reference point with respect to CCCFGS [14]. We note that a direct comparison is difficult, since (i) CCCFGS targets on a modified CKKS scheme, while we aim for standard RNS-based HE schemes; (ii) CCCFGS only reports an *estimated* prover time (25.3s) based on a loose upper bound for the following scenario: a ciphertext-ciphertext multiplication followed by a mod-switching and noise flooding, with CKKS parameters $N = 2^{13}$ and $L = 1$ (i.e., 3 primes as the HE modulus). For the reader’s reference, we evaluate our non-ZK version of PRISM for standard CKKS on the same scenario, and our prover time is 3.3s using a single thread.

8.4 Scalability of PRISM

For fixed HE and security parameters, PRISM’s total cost grows with the number and size of the polynomial relations induced by the HE circuit. The PIOPs have linear prover time, while the MF-PCS introduces quasi-linear prover time. Consequently, the online prover time grows quasi-linearly with the HE circuit size. Using similar arguments, we conclude that the offline time also grows quasi-linearly with the HE circuit size, while the online verifier time and proof size are sublinear in the HE circuit size.

We have evaluated PRISM in various scenarios with increasing workloads (cf. Table 2), and the results in Table 3 show that the concrete performance of PRISM is consistent with the above analysis. This indicates that PRISM exhibits scalability.

Table 3: Comparison of PRISM (ZK version) with ZHE [62]. The better result in each metric is highlighted in **bold.**

Scenario	Scheme	N	L	HE time (ms)	Offline time (s)		Prover time (s)		Verifier time (s)		Proof size (MB)	
					ZHE	PRISM	ZHE	PRISM	ZHE	PRISM	ZHE	PRISM
CM	CKKS	2 ¹³	2	3.45	1.02	1.55	0.96	3.39	0.77	0.13	6.30	1.14
	BGV	2 ¹³	2	4.17	1.56	1.92	1.49	4.84	1.21	0.19	8.58	1.74
MVM	CKKS	2 ¹³	2	36.88	14.32	19.43	13.46	46.68	11.28	1.78	71.95	16.15
	BGV	2 ¹³	2	43.54	22.44	25.31	21.18	68.22	17.71	2.74	106.17	25.05
LR	CKKS	2 ¹⁴	6	179.17	60.46	101.63	55.03	191.05	45.54	6.07	341.44	46.16
CA	CKKS	2 ¹⁴	7	554.65	186.30	326.99	173.16	572.92	145.02	18.69	1103.91	142.20

Table 4: Comparison of PRISM (non-ZK version) with ABPS [4]. The better result is highlighted in **bold.**

	ABPS	PRISM
Offline time (s)	1698.59	0.01
Prover time (s)	97.41	9.64
Verifier time (s)	0.45	0.17
Proof size (MB) [†]	0.60	3.30

[†] For ABPS, the proof size includes both the SNARK proof and the intermediate ciphertexts that must be transmitted to the verifier as inputs.

9 Conclusion

We presented PRISM, a practical zkSNARK for standard RNS-based HE schemes. Our key technical contributions are threefold: (1) MF-PCS, a new cryptographic primitive that can efficiently prove the cross-field modulo operations; (2) a novel PIOP for (I)NTT relations that achieves $O(\log N)$ verifier time while maintaining $O(N)$ prover time in a model with an offline phase; and (3) a VOLE-based technique that upgrades both (MF-)PCS and SumCheck-related PIOPs to zero-knowledge with small overhead. We conduct extensive experiments to demonstrate the efficiency of PRISM. The evaluation results show that PRISM achieves a favorable balance among prover time, verifier time, and proof size compared to prior works.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and suggestions. We also thank Yuanju Wei for his helpful discussions on this work.

References

- [1] 2024. Lattigo v6. Online: <https://github.com/tuneinsight/lattigo>. EPFL-LDS, Tune Insight SA.
- [2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Liger: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 2017*.
- [3] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. 2025. HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical. In *ASIACRYPT 2024*.
- [4] Shahla Atapoor, Karim Bagheri, Hilder V. L. Pereira, and Jannik Spiessens. 2024. Verifiable FHE via Lattice-based SNARKs. *IACR Communications in Cryptology* (2024).
- [5] Thomas Attema, Serge Fehr, and Michael Klooß. 2022. Fiat-Shamir Transformation of Multi-round Interactive Proofs. In *Theory of Cryptography: 20th International Conference*.
- [6] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. *Cryptology ePrint Archive*, Paper 2022/915. <https://eprint.iacr.org/2022/915>
- [7] Suyash Bagad, Quang Dao, Yuval Domb, and Justin Thaler. 2025. Speeding Up Sum-Check Proving. *Cryptology ePrint Archive*, Paper 2025/1117. <https://eprint.iacr.org/2025/1117>
- [8] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. 2023. Proximity gaps for Reed-Solomon codes. *J. ACM* 70, 5 (2023), 1–57.
- [9] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011*.
- [10] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. 2021. Flexible and Efficient Verifiable Computation on Encrypted Data. In *PKC 2021*.
- [11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing Vector OLE. In *ACM CCS 2018*.
- [12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM TOCT* (2014).
- [13] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK Compilers. In *EUROCRYPT 2020*.
- [14] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. 2025. Verifiable Computation for Approximate Homomorphic Encryption Schemes. *CRYPTO 2025*.
- [15] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2024. VERITAS: Plaintext Encoders for Practical Verifiable Homomorphic Encryption. In *ACM CCS 2024*.
- [16] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *EUROCRYPT 2023*.
- [17] Jung Hee Cheon, Kyoohyun Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A full RNS variant of approximate homomorphic encryption. In *SAC 2018*.
- [18] Jung Hee Cheon and Daehyun Jang. 2026. Cryptanalysis on Lightweight Verifiable Homomorphic Encryption. In *Advances in Cryptology – ASIACRYPT 2025*.
- [19] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017*.
- [20] Jung Hee Cheon, Dongwoo Kim, and Duhyeon Kim. 2020. Efficient homomorphic comparison methods with optimal complexity. In *ASIACRYPT 2020*.
- [21] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *EUROCRYPT 2020*.
- [22] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* (2020).
- [23] Benjamin E Diamond and Jim Posen. 2024. Proximity Testing with Logarithmic Randomness. *IACR Communications in Cryptology* (2024).
- [24] Benjamin E Diamond and Jim Posen. 2025. Succinct arguments over towers of binary fields. In *EUROCRYPT 2025*.
- [25] Changchang Ding and Yan Huang. 2025. Phedda: Post-Quantum Transparent zkSNARKs from Improved Polynomial Commitment and VOLE-in-the-Head with Application in Publicly Verifiable AES. In *IEEE S&P 2025*.

- [26] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive*, Paper 2022/1763. <https://eprint.iacr.org/2022/1763>
- [27] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*, Report 2012/144.
- [28] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO 1986*.
- [29] Dario Fiore, Anca Nitulescu, and David Pointcheval. 2020. Boosting Verifiable Computation on Encrypted Data. In *PKC 2020*.
- [30] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. 2024. How to Prove Statements Obviously?. In *CRYPTO 2024*.
- [31] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *CRYPTO 2010*.
- [32] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCs. In *EUROCRYPT 2013*.
- [33] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *ACM STOC 2009*.
- [34] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *ACM STOC 1985*.
- [35] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. 2023. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO 2023*.
- [36] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT 2016*.
- [37] Shai Halevi and Victor Shoup. 2014. Algorithms in HELIB. In *CRYPTO 2014*.
- [38] Shai Halevi and Victor Shoup. 2020. Design and implementation of HELIB: a homomorphic encryption library. *Cryptology ePrint Archive*, Paper 2020/1481. <https://eprint.iacr.org/2020/1481>
- [39] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics* (2018).
- [40] Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. 2020. Semi-parallel logistic regression for GWAS on encrypted data. *BMC medical genomics* (2020).
- [41] Fuchun Lin, Chaoping Xing, and Yizhou Yao. 2025. Interactive Line-Point Zero-Knowledge with Sublinear Communication and Linear Computation. In *ASIACRYPT 2024*.
- [42] Fengrun Liu, Haofei Liang, Tianyu Zhang, Yuncong Hu, Xiang Xie, Haisheng Tan, and Yu Yu. 2026. HasteBoots: Proving FHE Bootstrapping in Seconds. In *USENIX Security 2026*. <https://eprint.iacr.org/2025/261>
- [43] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *J. ACM* (1992).
- [44] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. 2012. A New Approach to Practical Active-Secure Two-Party Computation. In *CRYPTO 2012*.
- [45] Kabir Peshawaria, Zeyu Liu, Ben Fisch, and Eran Tromer. 2025. Laminar: Succinct SIMD-Friendly Verifiable FHE. *Cryptology ePrint Archive*, Paper 2025/2285. <https://eprint.iacr.org/2025/2285>
- [46] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. 2023. Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN. In *CRYPTO 2023*.
- [47] SEAL 2023. Microsoft SEAL (release 4.1). <https://github.com/Microsoft/SEAL>. Microsoft Research, Redmond, WA..
- [48] Srinath Setty, Justin Thaler, and Riad Wahby. 2024. Unlocking the lookup singularity with lasso. In *EUROCRYPT 2024*.
- [49] Alireza Shirzad, Sriram Sridhar, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2026. Relaxed Modular PCS from Arbitrary PCS and Applications to SNARKs for Integers. *Cryptology ePrint Archive*, Paper 2026/347. <https://eprint.iacr.org/2026/347>
- [50] Victor Shoup. 2021. NTL: a library for doing number theory (release 11.5.1). <https://github.com/libntl>.
- [51] Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* (2014).
- [52] Louis Tremblay Thibault and Michael Walter. 2025. Towards Verifiable FHE in Practice: Proving Correct Execution of TFHE’s Bootstrapping using plonky2. *ACM CCS 2025*.
- [53] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. 2023. Verifiable Fully Homomorphic Encryption. arXiv:2301.07041 [cs.CR]
- [54] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. <https://github.com/emp-toolkit>.
- [55] Yuanju Wei, Kaixuan Wang, Binwu Xiang, Xinxuan Zhang, Yi Deng, Xudong Zhu, Hailong Wang, Li Lin, and Lei Wang. 2025. Packed Sumcheck over Fields of Small Characteristic. *Cryptology ePrint Archive*, Paper 2025/719. <https://eprint.iacr.org/2025/719>
- [56] Yuanju Wei, Zhelei Zhou, Xinxuan Zhang, Songyu Wu, Binwu Xiang, Cheng Hong, and Yi Deng. 2026. GlueLUT: Generalized Lookup Table Arguments over Residue Rings via Auxiliary Fields. *Cryptology ePrint Archive*, Paper 2026/494. <https://eprint.iacr.org/2026/494>
- [57] 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 *IEEE S&P 2021*.
- [58] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *CRYPTO 2019*.
- [59] Tiacheng Xie, Yupeng Zhang, and Dawn Song. 2022. Orion: Zero Knowledge Proof with Linear Prover Time. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference*.
- [60] 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 *ACM CCS 2021*.
- [61] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. 2025. Phalanx: An FHE-Friendly SNARK for Verifiable Computation on Encrypted Data. In *ACM CCS 2025*.
- [62] Zhelei Zhou, Yun Li, Yuchen Wang, Zhaomin Yang, Bingsheng Zhang, Cheng Hong, Tao Wei, and Wenguang Chen. 2025. ZHE: Efficient Zero-Knowledge Proofs for HE Evaluations. In *IEEE S&P 2025*. <https://eprint.iacr.org/2025/770>
- [63] Vincent Zucca. 2018. *Towards efficient arithmetic for Ring-LWE based homomorphic encryption*. Ph. D. Dissertation. Sorbonne Université.

A Ethical Considerations

Stakeholders. Direct stakeholders include the research team and organizations integrating PRISM into verifiable or privacy-preserving systems. Indirect stakeholders include users whose data are processed by HE-based services, regulators and auditors. This research involves no human subjects, private data, or experiments on live systems, and authorship reflects each member’s contribution.

Benefits and public interest. PRISM improves the integrity of privacy-preserving computation by enabling verification of HE evaluations. Its techniques may also facilitate future research on PCS and zkSNARKs. Open-source release and documentation promote transparency and broader access to these results.

Potential harms and mitigations. Deployment requires implementation effort and independent security review. Residual risks include misuse for concealing illicit computations and privacy or correctness failures caused by improper implementation. We mitigate these risks by documenting security assumptions, recommended parameters, performance results, and known limitations. Production deployments should additionally undergo independent audits and comply with applicable legal and governance requirements.

Decision to conduct and publish. After evaluating the potential dual-use risks, we determined that the expected benefits to verifiable and privacy-preserving computation outweigh the potential harms when appropriate safeguards are applied. Publication enables independent scrutiny and supports further cryptographic research.

B Open Science

We provide the source code of PRISM at <https://github.com/antcplab/openprism>.

C Generative AI Usage

Generative AI tools (GPT, Claude, and Kimi) were used during the preparation of this work, specifically for assisting with code implementation, $\LaTeX$ formatting, and polishing of the writings. These tools were *not* used to generate any technical ideas, security proofs, or experimental results. All outputs produced by AI tools were carefully reviewed, verified, and, where necessary, revised by the authors. The authors take full responsibility for the writing and the results reported in this work.

D More Preliminaries

D.1 VOLE Functionality

In Figure 12, we provide the formal description of VOLE functionality $\mathcal{F}_{\text{VOLE}}$.

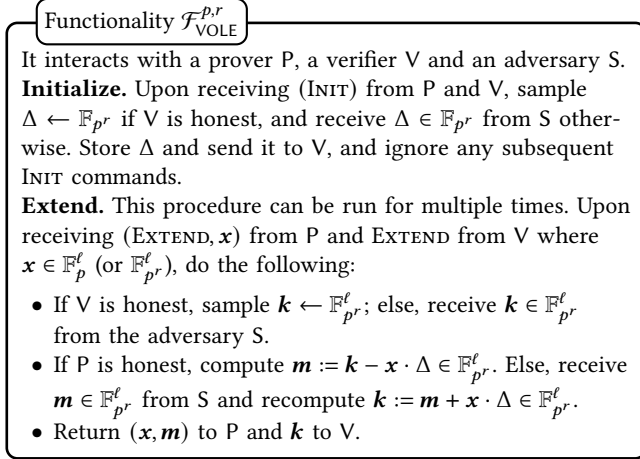


Figure 12: Functionality for VOLE

D.2 Formal Definition of (zk)SNARK

We provide the formal definition of (zk)SNARK below.

DEFINITION 6 ((zk)SNARK). We say a tuple of PPT algorithms (Setup, Prove, Verify) is a SNARK for $\mathcal{R}$ if the following holds.

- **Completeness.** For any $(\mathbf{x}; \mathbf{w}) \in \mathcal{R}$, we have

$$\Pr \left[(\mathbf{pk}, \mathbf{vk}) \leftarrow \text{Setup}(\lambda); \text{Verify}(\mathbf{vk}, \mathbf{x}, \pi) = 1 \right] \geq 1 - \text{negl}(\lambda).$$

- **Knowledge Soundness.** For all PPT adversary A, there exists a PPT extractor Ext such that for all randomness ρ , we have

$$\Pr \left[(\mathbf{pk}, \mathbf{vk}) \leftarrow \text{Setup}(\lambda); \begin{array}{l} \mathbf{w} \leftarrow \text{Ext}(\mathbf{pk}, \mathbf{vk}, \rho) \\ (\mathbf{x}, \pi) \leftarrow A(\mathbf{pk}; \rho) \end{array} : \wedge \text{Verify}(\mathbf{vk}, \mathbf{x}, \pi) = 1 \right] \leq \text{negl}(\lambda).$$

- **Succinctness.** The proof size $|\pi|$ and the time complexity of Verify are sublinear in $|\mathbf{x}|$.³

We say a SNARK is a zkSNARK if it additionally satisfies the following property:

- **Zero-Knowledge.** For any $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, there exists a stateful PPT simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$ such that

$$\begin{aligned} & \{(\mathbf{vk}, \pi) \mid (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{Setup}(\lambda); \pi \leftarrow \text{Prove}(\mathbf{pk}, \mathbf{x}, \mathbf{w})\} \\ & \stackrel{c}{\approx} \{(\mathbf{vk}, \pi) \mid (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{Sim}_1(\lambda; \rho); \pi \leftarrow \text{Sim}_2(\mathbf{x}, \mathbf{w})\}. \end{aligned}$$

³In this paper, we follow the succinctness definition of [35]. In other works (e.g., [16]), they may require the proof size and the verification time to be polylogarithmic in the statement size.

D.3 Polynomial Commitment Scheme

Here we provide the definition of PCS which is adapted from [35]. A PCS consists of the following algorithms:

- $(\mathbf{pk}, \mathbf{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu)$: It takes the security parameter λ and the number of variables in a polynomial μ , and outputs the prover key $\mathbf{pk}$ and the verifier key $\mathbf{vk}$.
- $(\text{com}, h) \leftarrow \text{Commit}(\mathbf{pk}, f)$: It takes the prover key $\mathbf{pk}$ and a polynomial $f \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ as inputs, and it outputs a commitment com , the opening hint h .
- $b \leftarrow \text{Open}(\mathbf{vk}, \text{com}, f, h)$: It takes the verifier key $\mathbf{vk}$, the commitment com , a polynomial $f \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ and the opening hint h as inputs, and it outputs a bit b indicating if f is the correct opening of com .
- $b \leftarrow \text{Eval}(\mathbf{pk}, \mathbf{vk}, \text{com}, f, \mathbf{x}, y)$: It is a protocol between a PPT prover P and verifier V. Both P and V holds com , $\mathbf{x} \in \mathbb{F}_{p^r}^\mu$, $y \in \mathbb{F}_{p^r}$. P additionally holds $f \in \mathcal{F}_{\mu, p}^{(\leq 1)}$ and $\mathbf{pk}$, while V additionally holds $\mathbf{vk}$. P attempts to convince V that $f(\mathbf{x}) = y^{(i)}$. Finally, V outputs a bit b indicating acceptance or rejection.

DEFINITION 7 ((zk) PCS). We say a tuple of protocols (PCSetup, Commit, Open, Eval) is an extractable PCS for multilinear polynomials if the following conditions hold:

- **Completeness:** For any multilinear polynomial over the master field $f \in \mathbb{F}_p[\mu]$, the following probability is at least $1 - \text{negl}(\lambda)$

$$\Pr \left[\begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu); \\ (\text{com}, h) \leftarrow \text{Commit}(\mathbf{pk}, f) \end{array} : \begin{array}{l} \text{Eval}(\mathbf{pk}, \mathbf{vk}, \text{com}, f, \mathbf{x}, y) = 1 \\ \wedge f(\mathbf{x}) = y \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

- **Binding:** For any PPT adversary A, size parameter $\mu \geq 1$, we have

$$\Pr \left[\begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu); \\ (\text{com}, f_0, h_0, f_1, h_1) \leftarrow A(\mathbf{pk}); \\ b_0 \leftarrow \text{Open}(\mathbf{vk}, \text{com}, f_0, h_0); \\ b_1 \leftarrow \text{Open}(\mathbf{vk}, \text{com}, f_1, h_1) \end{array} : \begin{array}{l} b_0 = b_1 \neq 0 \\ \wedge f_0 \neq f_1 \end{array} \right] \leq \text{negl}(\lambda).$$

- **Knowledge soundness:** Given $(\mathbf{pk}, \mathbf{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu)$, Eval indicates a SNARK for the following NP relation:

$$\mathcal{R}'_{\text{KS}} = \left\{ \left(\begin{array}{l} (\text{com}, \mathbf{x}, y); \\ (f, h) \end{array} \right) : \begin{array}{l} \text{Open}(\mathbf{vk}, \text{com}, f, h) = 1 \\ \wedge f(\mathbf{x}) = y \end{array} \right\}.$$

Moreover, we say a PCS is **zero-knowledge** if (i) com reveals nothing about f , i.e., there exists a stateful PPT simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$ such that $\{(\mathbf{vk}, \text{com}) \mid (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{PCSetup}(1^\lambda, \mu); (\text{com}, h) \leftarrow \text{Commit}(\mathbf{pk}, f)\} \stackrel{c}{\approx} \{(\mathbf{vk}, \text{com}) \mid (\mathbf{pk}, \mathbf{vk}) \leftarrow \text{Sim}_1(1^\lambda, \mu; \rho); \text{com} \leftarrow \text{Sim}_2(\rho)\}$; (ii) Eval indicates a zkSNARK for $\mathcal{R}'_{\text{KS}}$.

D.4 Polynomial IOP

To define a PIOP, we have to introduce the notion of interactive proofs of knowledge, which is adapted from [16].

DEFINITION 8 (INTERACTIVE PROOF/ARGUMENT OF KNOWLEDGE). An interactive protocol $\Pi = (G, I, P, V)$ between a prover P and a verifier V is an interactive argument of knowledge for an indexed relation $\mathcal{R}$ if the following properties hold:

- **Perfect completeness:** For any $(\mathbf{i}, \mathbf{x}, \mathbf{w}) \in \mathcal{R}$, we have

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow G(1^\lambda); \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow I(\text{pp}, \mathbf{i}) \end{array} : \langle P(\mathbf{pk}, \mathbf{x}, \mathbf{w}), V(\mathbf{vk}, \mathbf{x}) \rangle = 1 \right] = 1.$$

- **Soundness:** Let $\mathcal{L}$ be the language corresponding to $\mathcal{R}$ such that $\mathbb{x} \in \mathcal{L}$ if and only if there exists a $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. We say Π is computationally sound, if for any $x \notin \mathcal{L}$ and any PPT adversary $A = (A_1, A_2)$, the following is negligible in λ :

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow G(1^\lambda); \\ (\mathbb{i}, \mathbb{x}, \text{st}) \leftarrow A_1(\text{pp}); : \langle A_2(\mathbb{i}, \mathbb{x}, \text{st}), V(\text{vk}, \mathbb{x}) \rangle = 1 \\ (\text{pk}, \text{vk}) \leftarrow I(\text{pp}, \mathbb{i}) \end{array} \right]$$

- **Knowledge soundness:** We say Π is knowledge sound (or an argument of knowledge), if for any PPT adversary $A = (A_1, A_2)$, there exists a PPT extractor Ext such that given oracle access to A , the following probability is negligible in λ :

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow G(1^\lambda); \\ (\mathbb{i}, \mathbb{x}, \text{st}) \leftarrow A_1(\text{pp}); : \text{w} \leftarrow \text{Ext}^A(\text{pp}, \mathbb{i}, \mathbb{x}) \\ (\text{pk}, \text{vk}) \leftarrow I(\text{pp}, \mathbb{i}); : \wedge (\mathbb{i}, \mathbb{x}, \text{w}) \notin \mathcal{R} \\ \langle A_2(\mathbb{i}, \mathbb{x}, \text{st}), V(\text{vk}, \mathbb{x}) \rangle = 1 \end{array} \right]$$

We note that, if A is unbounded, then Π is called an interactive proof of knowledge.

An interactive protocol can also satisfy the following properties.

- **Public-coin.** Π is said to be public-coin, if all of the verifier messages can be computed as a deterministic function given a random public input.
- **Zero-knowledge.** Π is said to be zero-knowledge, if for any auxiliary input $z \in \{0, 1\}^{\text{poly}(\lambda)}$ and any PPT adversary $A = (A_1, A_2)$, there exists a PPT simulator Sim such that the following is negligible in λ :

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow G(1^\lambda); \\ (\mathbb{i}, \mathbb{x}, \text{w}, \text{st}) \leftarrow A_1(z, \text{pp}); : \langle P(\mathbb{i}, \mathbb{x}, \text{w}), A_2(\text{st}, \mathbb{i}, \mathbb{x}) \rangle = 1 \\ (\text{pk}, \text{vk}) \leftarrow I(\text{pp}, \mathbb{i}); : \wedge (\mathbb{i}, \mathbb{x}, \text{w}) \in \mathcal{R} \end{array} \right]$$

$$-\Pr \left[\begin{array}{l} (\text{pp}, \sigma) \leftarrow \text{Sim}(1^\lambda); \\ (\mathbb{i}, \mathbb{x}, \text{w}, \text{st}) \leftarrow A_1(z, \text{pp}); : \langle \text{Sim}(\sigma, z, \text{pk}, \mathbb{i}), A_2(\text{st}, \mathbb{i}, \mathbb{x}) \rangle = 1 \\ (\text{pk}, \text{vk}) \leftarrow I(\text{pp}, \mathbb{i}); : \wedge (\mathbb{i}, \mathbb{x}, \text{w}) \in \mathcal{R} \end{array} \right]$$

Now we can define PIOP as follows, which is adapted from [16].

DEFINITION 9 (PIOP). A PIOP is a public-coin interactive proof of knowledge for a polynomial oracle relation $\mathcal{R} = \{(\mathbb{i}, \mathbb{x}, \mathbb{w})\}$. The relation is an oracle relation in that $\mathbb{i}$ and $\mathbb{x}$ can contain oracles to μ -variable polynomials over some field $\mathbb{F}_p$. The oracles specify μ and the degree in each variable. These oracles can be queried at arbitrary points in $\mathbb{F}_p^\mu$ to evaluate the polynomial at these points. In every protocol message, P sends multi-variate polynomial oracles to V , who sends a random challenge back to V .

D.5 More Details about RNS-based HE

Our zkSNARK is applicable to all RLWE-based HEs including BFV [27], BGV [12] and CKKS [19]. Here we take CKKS as an example to describe HE. Notice that, a CKKS plaintext $m \in R$ encodes a vector $\mathbf{a} \in \mathbb{C}^{\frac{N}{2}}$, and m can be decoded into $\mathbf{a}$. For simplicity, we omit the encoding/decoding algorithms here (ref. [19]), and describe the rest algorithms as follows.

- **ParamGen($1^\lambda, q, L, \eta$):** Given the security parameter λ , a base module q , a maximum level L of computation, the precision η , the algorithm generates the parameters:
 - Choose $L + 2$ primes $\mathcal{D} := \{p_0, q_0, q_1, \dots, q_L\}$. We write $Q_\ell := \prod_{i=0}^\ell q_i$ for $\ell \in [0, L]$.

- Choose a power-of-two integer N .
- Choose small distributions for secret key/encryption key/error $\chi_{\text{key}}/\chi_{\text{enc}}/\chi_{\text{err}}$ over R respectively.

Finally, Output params $:= (\mathcal{D}, N, \chi_{\text{key}}, \chi_{\text{enc}}, \chi_{\text{err}})$ which is a default input to the following algorithms.

- **KSGen(s_1, s_2):** Given $s_1, s_2 \in R$, sample $\tilde{a}_i \leftarrow R_{p_0 Q_L}$ and $\tilde{e}_i \leftarrow \chi_{\text{err}}$ for each $i \in [0, L]$. Output switching keys $\text{swk} = \{\text{swk}_i = (\tilde{b}_i, \tilde{a}_i)\}_{i \in [0, L]}$, where $\tilde{b}_i := -\tilde{a}_i \cdot s_2 + \tilde{e}_i + p_0 \cdot B_i \cdot s_1 \in \mathbb{Z}_{p_0 Q_L}$ for integer $B_i \in \mathbb{Z}_{Q_L}$ s.t. $B_i = 1 \bmod q_i$ and $B_i = 0 \bmod q_j$ for all $j \neq i$.
- **KeyGen(1^λ):** Given the security parameter λ , the algorithm samples the following keys:
 - Sample $s \leftarrow \chi_{\text{key}}$, and set secret key as $\text{sk} := (1, s)$.
 - Sample $a \leftarrow R_{Q_L}$ and $e \leftarrow \chi_{\text{err}}$, and set the public key as $\text{pk} := (b, a)$, where $b := -a \cdot s + e \bmod Q_L$.
 - Compute the relinearization key $\text{rlk} \leftarrow \text{KSGen}(s^2, s)$.
 - Compute the rotation key $\text{rk}^{(k)} \leftarrow \text{KSGen}(s^{(k)}, s)$ for each rotation step $k \in [1, \frac{N}{2} - 1]$, where $s^{(k)}$ is the rotated secret with respect to s .
- Finally, output $\text{pk}, \text{sk}, \text{rlk}, \text{rk} := \{\text{rk}^{(k)}\}_{k \in [1, \frac{N}{2} - 1]}$.
- **Enc(pk, m):** Given the public key pk and the plaintext $m \in R_{Q_0}$, sample $v \leftarrow \chi_{\text{enc}}$ and $e_0, e_1 \leftarrow \chi_{\text{err}}$. Output the ciphertext $\text{ct} := v \cdot \text{pk} + (m + e_0, e_1) \in \mathcal{R}_{Q_L}^2$. We will abuse the notion of encryption so that if $v \in \mathbb{C}^{\frac{N}{2}}$ is encoded by m then we will also say ct encrypts v .
- **Dec(sk, ct):** Given secret key sk and a ciphertext $\text{ct} \in \mathcal{R}_{Q_\ell}^2$, output the plaintext $m = \langle \text{ct}, \text{sk} \rangle \bmod Q_0 \in R_{Q_0}$.
- **Add(ct, ct'):** Given two ciphertext $\text{ct}, \text{ct}' \in \mathcal{R}_{Q_\ell}^2$, output the ciphertext $\text{ct}_{\text{Add}} := \text{ct} + \text{ct}' \in \mathcal{R}_{Q_\ell}^2$.
- **ModSwitch($\text{ct}, Q_\ell, Q_{\ell-1}$):** Given the ciphertext $\text{ct} \in \mathcal{R}_{Q_\ell}^2$ and two modulus $Q_\ell, Q_{\ell-1}$, output the ciphertext $\text{ct}' := \lfloor (Q_\ell / Q_{\ell-1})^{-1} \cdot \text{ct} \rfloor \bmod Q_{\ell-1}$.
- **KeySwitch(swk, ct)**⁴ Given the switching key $\text{swk} = \{\text{swk}_i\}_{i \in [0, L]}$ and a ciphertext $\text{ct} = (c_0, c_1) \in \mathcal{R}_{Q_\ell}^2$, the algorithm performs the following:
 - Let $c_{1,i} = c_1 \bmod q_i$ for $i \in [0, \ell]$.
 - Compute $\tilde{\text{ct}} := \sum_{i=0}^\ell c_{1,i} \cdot \text{swk}_i \bmod p_0 Q_\ell$.
 - Output $\text{ct}' := (c_0, 0) + \text{ModSwitch}(\tilde{\text{ct}}, p_0 Q_\ell, Q_\ell)$.
- **Mult($\text{rlk}, \text{ct}, \text{ct}'$):** Given the relinearization key rlk and two ciphertexts $\text{ct} = (c_0, c_1), \text{ct}' = (c'_0, c'_1) \in \mathcal{R}_{Q_\ell}^2$, the algorithm performs the following:
 - Compute $d_0 := c_0 c'_0, d_1 := c_0 c'_1 + c'_0 c_1, d_2 := c_1 c'_1 \bmod Q_\ell$.
 - Invoke $\tilde{\text{ct}}' := \text{KeySwitch}(\text{rlk}, (0, d_2))$.
 - Output $\text{ct}_{\text{Mult}} := (d_0, d_1) + \tilde{\text{ct}}' \bmod Q_\ell$.
- **Rotate($\text{rk}^{(k)}, \text{ct}$):** Given the rotation key $\text{rk}^{(k)}$ and the ciphertext $\text{ct} \in \mathcal{R}_{Q_\ell}^2$, it performs the following:
 - Rotate the ciphertext $\text{ct} = (b, a)$ and obtain $(b^{(k)}, a^{(k)})$.
 - Output $\text{ct}' := \text{KeySwitch}(\text{rk}^{(k)}, (b^{(k)}, a^{(k)}))$.

Both modulus-switching and key-switching algorithms are crucial, since they can be used to build two important evaluation algorithms: ciphertext-ciphertext multiplication (denoted as Mult) and rotation (denoted as Rotate). Given ciphertexts ct_a, ct_b which

⁴We consider the key-switching method and the corresponding switching key [17, 39] that are implemented in Microsoft SEAL library [47].

encrypt $\mathbf{a}$ and $\mathbf{b}$ respectively, Mult outputs a ciphertext ct_c which encrypts $\mathbf{c} = \mathbf{a} \odot \mathbf{b}$. Given a ciphertext ct which encrypts $\mathbf{a} = (a_1, \dots, a_{N/2})$, Rotate outputs a ciphertext ct' which encrypts a shifted vector $(a_{k+1}, \dots, a_{N/2}, a_1, \dots, a_k)$, where k is a parameter.

D.6 Connections between Cross-Field Modulo Operations, (I)NTTs and vHEs

In the following, we mainly discuss the cross-field modulo operations and (I)NTT operations in RLWE-based HE.

Cross-field modulo operation. To better illustrate why RNS-based HE requires cross-field modulo operations, here we follow [4, 63] and start by introducing a function called FastBconv [4, 63], which converts a value $c \in R_{Q_\ell}$ in a RNS basis $Q_\ell = q_1 \cdots q_\ell$ into another basis $\tilde{Q}_k = \tilde{q}_1 \cdots \tilde{q}_k$:

$$\text{FastBconv}(c, Q_\ell, \tilde{Q}_k) = \left(\sum_{i \in [\ell]} [c \cdot (Q_\ell/q_i)^{-1}]_{q_i} \cdot (Q_\ell/q_i) \bmod \tilde{q}_j \right)_{j \in [k]},$$

where $[x]_{q_i}$ denotes $x \bmod q_i$. It is easy to see that we can use cross-field modulo operations to compute FastBconv: For each $i \in [\ell]$, we need to take a value $y_i := [c \cdot (Q_\ell/q_i)^{-1}]_{q_i}$ over $\mathbb{F}_{q_i}$ as input, then compute $y_i \cdot (Q_\ell/q_i) \bmod \tilde{q}_j$, which is equivalent to $(y_i \bmod \tilde{q}_j) \cdot (Q_\ell/q_i \bmod \tilde{q}_j) \bmod \tilde{q}_j$, for all $j \in [k]$. Since $y_i \in \mathbb{F}_{q_i}$, the computation of $y_i \bmod \tilde{q}_j$ is exactly a cross-field modulo operation. Therefore, we conclude that we can use cross-field modulo operations to compute FastBconv.

Now we take CKKS as an example, and show how to use FastBconv to describe key-switching and mod-switching. Recall that, we have already shown in Section 1 that the CKKS mod-switching requires computing $(\text{ct} \bmod q_\ell) \bmod Q_{\ell-1}$, which is equivalent to computing $\text{FastBconv}([\text{ct}]_{q_\ell}, q_\ell, Q_{\ell-1})$. As for CKKS key-switching, given a ciphertext $(c_0, c_1) \in R_{Q_\ell}^2$ and the switching key $\text{swk} \in R_{p_0 Q_\ell}^2$, the algorithm needs to switch the modulus of c_1 from Q_ℓ to $p_0 Q_\ell$ by computing $\tilde{c}'_1 = \text{FastBconv}(c_1, Q_\ell, p_0 Q_\ell)$ first, then we are able to multiply $\tilde{c}'_1$ with swk over $\mathbb{Z}_{p_0 Q_\ell}$. Since both mod-switching and key-switching require FastBconv, we conclude that we can use the cross-field modulo operations to implement RNS-based HE evaluation algorithms. We also note that, the HE implementations of [4] are similar to ours.

Connection to vHE. We take $\text{FastBconv}([c]_{q_\ell}, q_\ell, Q_{\ell-1})$ as an example, and show how to prove it using our MF-PCS schemes. First, by computing $[c]_{q_\ell}$, we can obtain a multilinear polynomial f over $\mathbb{F}_{q_\ell}$. Then, we select r_i such that $q_i^{-r_i} \leq 2^{-\lambda}$ for each $i \in [\ell]$. Finally, we fix $\mathbb{F}_{q_\ell}$ as the master field, and let P invoke MF-PCS.Commit on f and $\{q_i, r_i\}_{i \in [\ell]}$. When V requires polynomial evaluations, P can simply run MF-PCS.VerCom and MF-PCS.Eval with V.

(I)NTT operation. As discussed in [62], there are two forms of ciphertexts: NTT form and coefficient form. By using NTT form, additions and multiplications can be easily performed. Note that, we need to apply an inverse NTT to obtain the coefficient form of the ciphertext before the modulo operations; after the modulo operations, we apply an NTT to obtain the NTT form of the ciphertext. Therefore, there are multiple (I)NTT instances during the whole HE evaluations.

Connection to vHE. For better efficiency, we take the approach of the prior work [42, 55] to prove (I)NTT. Assume there are t (I)NTT instances $\{f_i, g_i\}_{i \in [t]}$ over $\mathbb{F}_p$, where f_i is in the coefficient form and

g_i is in the NTT form. We first let V send t random $s_1, \dots, s_t \leftarrow \mathbb{F}_{p^r}$ to P, where r is selected such that $p^{-r} \leq 2^{-\lambda}$. Then both the parties compute $\tilde{f} := \sum_{i \in [t]} s_i f_i$ and $\tilde{g} := \sum_{i \in [t]} s_i g_i$. Finally, we let both parties invoke the (I)NTT PIOP on $(\tilde{f}, \tilde{g})$.

E Microbenchmark of ZHE

To illustrate the cost of proving cross-field modulo operations, we profile ZHE [62] on the CM scenario described in Section 8.3 with $N = 2^{13}$ and $L = 2$ for both CKKS and BGV. We decompose the total proving cost into two parts: (i) the cost of proving cross-field modulo operations (denoted “Mod”), and (ii) the remaining cost. The results are shown in Table 5.

Table 5: Cost breakdown of ZHE [62] on the CM Scenario ($N = 2^{13}, L = 2$). “Mod” denotes the cost of proving cross-field modulo operations; “Ratio” denotes the fraction of the total cost attributed to modulo operations.

		Prover time (s)	Verifier time (s)	Proof size (MB)
CKKS	Mod	0.77	0.66	2.41
	Total	0.96	0.77	6.30
	Ratio	80.2%	85.7%	38.3%
BGV	Mod	1.28	1.08	4.20
	Total	1.49	1.21	8.58
	Ratio	85.8%	89.6%	49.0%

As shown in Table 5, proving the cross-field modulo operations accounts for over 80% of the total prover time and over 85% of the total verifier time for both CKKS and BGV. This confirms that the cross-field modulo operation is the dominant bottleneck in proving RNS-based HE evaluations.

F Detailed Analysis of Proof-Friendly CKKS

In this section, we provide a detailed analysis of the proof-friendly CKKS scheme proposed by Cascudo et al. [14] (hereafter CCCFGS) and discuss the design tradeoffs compared to standard RNS-based CKKS.

F.1 Overview of the Proof-Friendly CKKS

CCCFGS takes an interesting approach by redesigning the CKKS scheme so that all operations stay within a single fixed ring, thereby avoiding the need to prove cross-field consistency entirely. The scheme makes two key modifications to standard CKKS:

Modification 1: Incomplete NTT via extension-degree primes. Standard CKKS chooses RNS primes $p_i \equiv 1 \pmod{2N}$ so that $X^N + 1$ splits completely into N linear factors over $\mathbb{Z}_{p_i}$, enabling the standard NTT. CCCFGS instead chooses primes of the form $p_i = 2a \cdot (N/d) + 1$ for some odd integer a , so that $X^N + 1$ factors into N/d irreducible polynomials of degree d over $\mathbb{Z}_{p_i}$:

$$X^N + 1 = \prod_{j=0}^{N/d-1} (X^d - \zeta^{2j+1}),$$

where ζ is a primitive $(2N/d)$ -th root of unity modulo p_i . By the CRT, each R_{p_i} decomposes into N/d copies of the finite field $\mathbb{F}_{p_i^d}$. The parameter d (set to 2 or 4 in [14]) controls the size of the exceptional set $|S| = p_i^d$, which is used to achieve negligible soundness error in their proof system.

Since $X^N + 1$ no longer splits completely, the standard NTT cannot be used. Instead, CCCFGS employs an *incomplete NTT*: the full N -dimensional NTT is replaced by d independent (N/d) -dimensional negacyclic NTTs, followed by polynomial multiplication in the d -dimensional residue components.

Modification 2: All operations in a single ring. CCCFGS defines $z_\ell = \text{CRT}(1, \dots, 1, 0, \dots, 0) \in R_{q_0}$ and uses them to “manually”

switch levels within R_{q_0} . A level- ℓ ciphertext is represented as an element of R_{q_0} whose RNS components beyond index ℓ are zero. Maintenance operations, including key-switching and modulus-switching, are performed within R_{q_0} .

F.2 Design Tradeoffs

The CCCFGS approach offers a clean solution to the cross-field consistency problem by eliminating it at the HE design level. However, this comes with certain tradeoffs that are worth noting.

Reduced plaintext slots. Standard CKKS provides $N/2$ complex plaintext slots, while in proof-friendly CKKS, $X^N + 1$ splits into N/d factors of degree d , reducing the number of plaintext slots to $N/(2d)$. This means that, to process the same amount of data, the proof-friendly variant may require d times more ciphertexts, which could increase both the HE computation cost and the proving cost.

Different polynomial arithmetic. The incomplete NTT uses a different multiplication strategy in the d -dimensional residue components. According to the benchmarks reported in [14], a single polynomial multiplication takes 0.054 ms for $d = 4$ versus 0.013 ms for standard NTT (using Intel HEXL) at $N = 2^{13}$. At the level of a full CKKS multiplication (which involves multiple polynomial multiplications plus key switching), the difference is smaller: 1.205 ms for $d = 4$ versus 1.023 ms for standard CKKS at $N = 2^{13}$, $L = 3$.

Compatibility considerations. Applications built on standard CKKS libraries (e.g., Microsoft SEAL [47], HELib [37]) would need adaptation to use the proof-friendly variant.

Fixed ciphertext representation. In standard CKKS, modulus switching progressively reduces the ciphertext modulus, shrinking ciphertext size and accelerating lower-level operations. In the proof-friendly variant, ciphertexts are always represented in R_{q_0} regardless of the current level. While the high-level RNS components are logically zero and may not require active computation, the fixed-size representation may result in larger ciphertexts size when compared to the standard CKKS.

G Security Proofs

G.1 Proof of Theorem 1

PROOF. Completeness. Completeness holds by design.

Binding. We prove by contradiction. We assume there is a PPT adversary A who outputs a commitment $\text{com} = (c_U, c_T)$ and two pairs (f_0, h_0) and (f_1, h_1) such that $f_0 \neq f_1$ and $\text{Open}(\text{vk}, \text{com}, f_0, h_0) = \text{Open}(\text{vk}, \text{com}, f_1, h_1) = 1$. By the binding property of the Merkle

tree commitment, we know that $h_0 = h_1$. Parse h_0 as U and T and let $\hat{U}^{(0)}, \hat{T}^{(0)}$ (resp. $\hat{U}^{(1)}, \hat{T}^{(1)}$) be the corresponding message matrix and concatenated parity check matrix of f_0 (resp. f_1).

Notice that, if $f_0 \neq f_1$, then we must have $|\text{dist}(\hat{U}^{(0)}, \hat{U}^{(1)})| \geq 1$, which leads to $|\text{dist}(\hat{T}^{(0)}, \hat{T}^{(1)})| \geq d - 1$. However, since $\text{Open}(\text{vk}, \text{com}, f_0, h_0) = \text{Open}(\text{vk}, \text{com}, f_1, h_1) = 1$, we have

$$\begin{aligned} & |\text{dist}(\hat{U}^{(0)}, \hat{U}^{(1)})| + |\text{dist}(\hat{T}^{(0)}, \hat{T}^{(1)})| \\ & \leq (|\text{dist}(\hat{U}^{(0)}, U)| + |\text{dist}(\hat{T}^{(0)}, T)|) + \\ & \quad (|\text{dist}(\hat{U}^{(1)}, U)| + |\text{dist}(\hat{T}^{(1)}, T)|) \\ & < d/2 + d/2 = d, \end{aligned}$$

which leads to a contradiction. Therefore, we prove that $f_0 = f_1$, which shows that the binding property holds.

Knowledge soundness. In order to prove the knowledge soundness, we first prove the soundness, then prove the extractability. Soundness. Here we prove that if P can pass the testing phase, then with overwhelming probability that the event (1) for $i \in [n+1]$, the committed matrices $U^{(i)}, T^{(i)}$ are close to a codeword matrix $[\tilde{U}^{(i)}, \tilde{T}^{(i)}]$ in $C^{(i)}$ holds. Furthermore, if $\text{VerCom}(\text{pk}, \text{vk}, \text{com}, \{f^{(i)}\}_{i \in [n]}) = 1$, then with overwhelming probability that the following events hold: (2) for all $\mathbf{b} \in B_\mu$, $i \in [2, n]$: $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b}) \bmod p_i$; (3) for all $\mathbf{b} \in B_\mu$: $f^{(1)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(n+1)}(\mathbf{b})$.

To prove event (1), we recall a claim by Ben-Sasson et al. [8] in the following:

CLAIM 1. Let $C^{(i)}$ be a $[m', m, d]$ -RS code over $\mathbb{F}_{p_i}$. Let $\hat{U} = (\hat{U}_j)_{j \in [m]}$ $\in \mathbb{F}_{p_i}^{m \times m'}$, and for each $j \in [m]$ let c_j be the closest codeword in $C^{(i)}$ to $\hat{U}_j$. Let $\delta \in (0, \frac{1-\rho}{2}]$, where ρ denotes the rate. Let E be a subset of the columns $k \in [m']$ of $\hat{U}$ on which there is even one row $j \in [m]$ such that $c_{j,k} \neq \hat{U}_{j,k}$. Let $\epsilon = \frac{m'}{p_i}$. With probability at least $1 - \epsilon$ over the choice of $\mathbf{s} \in \mathbb{F}_{p_i}^m$, $\mathbf{s}^T \cdot \hat{U}$ has distance at least $|E|$ from any codeword in $C^{(i)}$.

Now we prove event (1) by contradiction. We first assume that $\text{dist}([U^{(i)}, T^{(i)}], C^{(i)}) > \delta \cdot m'$. For $i \in [n+1]$, $j \in [r_i]$, we recall that $\mathbf{w}^{(i,j)} := (\mathbf{s}^{(i,j)})^T \cdot U^{(i)}$. Since $\text{dist}([U^{(i)}, T^{(i)}], C^{(i)}) > \delta \cdot m'$, by Claim 1, we know that the probability over the verifier's choice of $\mathbf{s}^{(i,j)}$ that there exists a codeword $\mathbf{a}$ such that $\text{dist}(\mathbf{a}, \text{Enc}^{(i)}(\mathbf{w}^{(i,j)})) < \delta \cdot m'$ is ϵ . If no such $\mathbf{a}$ exists, we have $\text{dist}(\text{Enc}^{(i)}(\mathbf{w}^{(i,j)}), C^{(i)}) > \delta \cdot m'$, then the probability of passing step (3) of the testing phase is $(1 - \delta)^Y$. Notice that, for each $i \in [n+1]$, the step 1 is repeated for r_i times. Thus we conclude that, if the prover can pass the testing phase with probability at least $\max\{\epsilon^{r_i} + r_i \cdot (1 - \delta)^Y\}_{i \in [n+1]}$, then there exists a codeword matrix $[\tilde{U}^{(i)}, \tilde{T}^{(i)}]$ such that $\text{dist}([U^{(i)}, T^{(i)}], [\tilde{U}^{(i)}, \tilde{T}^{(i)}]) < \delta \cdot m' \leq \frac{d}{2}$ for each $i \in [n+1]$. In other words, $[U^{(i)}, T^{(i)}]$ can be uniquely decoded to $\tilde{U}^{(i)}$ since $\frac{d}{2}$ is the unique decoding bound of the RS code. Notice that, for each $i \in [n+1]$, the extended polynomial $f^{(i)}$ can be obtained from $\tilde{U}^{(i)}$.

Then we prove event (2) and (3) using the following arguments. Firstly, for each $i \in [n]$, we prove $\tilde{U}^{(i)} = \tilde{U}^{(n+1)} \bmod p_i$ by contradiction. Assuming there exists a $k \in [m]$ s.t. $\tilde{U}_k^{(i)} \neq \tilde{U}_k^{(n+1)} \bmod p_i$, then the probability that $(\mathbf{s}^{(i,j)})^T \cdot \tilde{U}^{(i)} = (\mathbf{s}^{(i,j)})^T \cdot \tilde{U}^{(n+1)} \bmod p_i$ is

at most p_i^{-1} . Since $s^{(i,j)}$ is sampled for r_i times, $\tilde{U}^{(i)} = \tilde{U}^{(n+1)} \bmod p_i$ holds with probability at least $1 - p_i^{-r_i}$. Secondly, we prove $\tilde{U}^{(1)}$ is identical to $\tilde{U}^{(n+1)}$ in the canonical integer representative by contradiction. If there exists a pair (j, k) such that $\tilde{U}_{j,k}^{(1)} \neq \tilde{U}_{j,k}^{(n+1)}$, then by the above argument, we must have $\tilde{U}_{j,k}^{(1)} + t \cdot p_1 = \tilde{U}_{j,k}^{(n+1)}$ where $t \geq 1$, which leads to $\tilde{U}_{j,k}^{(n+1)} \geq p_1$. However, by the knowledge soundness of the SNARK for $\mathcal{R}_{\text{Range}}$, from an accepting proof we can extract a polynomial $f_{\text{Range}}^{(n+1)}$ such that: $f_{\text{Range}}^{(n+1)}$ opens the auxiliary-field component of com; and $\forall \mathbf{b} \in B_\mu : f_{\text{Range}}^{(n+1)}(\mathbf{b}) \in [0, p_1 - 1]$. Independently, by the unique-decoding analysis above, the auxiliary-field component of the same commitment com uniquely decodes to $\tilde{U}^{(n+1)}$. The binding condition therefore implies that $\tilde{U}^{(n+1)}$ is the evaluation matrix of $f_{\text{Range}}^{(n+1)}$. Consequently, we have $\tilde{U}_{j,k}^{(n+1)} < p_1$, which contradicts $\tilde{U}_{j,k}^{(n+1)} \geq p_1$. Thus, we have proven that $\tilde{U}^{(1)}$ is identical to $\tilde{U}^{(n+1)}$, which implies the event (3). Finally, since we have already proven that $\tilde{U}^{(i)} = \tilde{U}^{(n+1)} \bmod p_i$, we can conclude that $\forall i \in [2, n] : \tilde{U}^{(i)} = \tilde{U}^{(1)} \bmod p_i$, which implies event (2).

Extractability. We prove the extractability by describing the extraction strategy of a straight-line extractor Ext. Firstly, Ext emulates the random oracle for the Merkle tree commitment; in this way, Ext is capable of knowing all the committed matrices $U^{(i)}, T^{(i)}$ for $i \in [n + 1]$. Secondly, by the soundness analysis, we know that $U^{(i)}, T^{(i)}$ can be uniquely decoded to $\tilde{U}^{(i)}$; therefore, Ext can invoke the efficient decoding algorithm of the RS code to extract $\tilde{U}^{(i)}$, which is equivalent to a polynomial $f^{(i)}$. Finally, also by the soundness analysis, we know that $f^{(i)}(\mathbf{b}) \stackrel{\mathbb{Z}}{=} f^{(1)}(\mathbf{b}) \bmod p_i$ for all $\mathbf{b} \in B_\mu$ and $i \in [2, n]$; thus, Ext can output $\{f^{(i)} \in \mathcal{F}_{\mu, p_i}^{(\leq 1)}\}_{i \in [n]}$ which satisfy the NP relation $\mathcal{R}_{\text{KS}}$. $\square$

G.2 Proof of Theorem 2

PROOF. Completeness. Completeness holds by design.

Knowledge soundness. As proven in [16, Lemma 2.3], any sound PIOPs are knowledge sound. Therefore, here we only need to prove the soundness of our PIOP.

In our construction, we check the NTT relation $\mathcal{R}_{\text{NTT}}$ by verifying if the following equation

$$\sum_{j \in [0, N-1]} s^j \cdot u_j = \sum_{j \in [0, N-1]} L_j(s) \cdot v_j \quad (1)$$

holds for a random $s \in \mathbb{F}_{p^r}$. By Schwartz-Zippel lemma, the NTT relation holds with at least $1 - p^{-r}$ probability if the equation 1 holds. Let the soundness error of ZeroCheck and SumCheck PIOPs be $\delta_{\text{Zero}}, \delta_{\text{Sum}}$, respectively. Then the equation 1 holds with at least $1 - \delta_{\text{Zero}} - \delta_{\text{Sum}}$ probability, if the verifier outputs acceptance in our NTT PIOP. In conclusion, by the union bound, the soundness error of our NTT PIOP is $\delta_{\text{Zero}} + \delta_{\text{Sum}} + p^{-r}$, which is negligible. Therefore, our NTT PIOP is (knowledge) sound. $\square$

G.3 Proof of Theorem 3

PROOF. Since the protocol in Figure 7 is based on the non-ZK construction in Figure 5, we first briefly argue the completeness

and knowledge soundness, then provide a detailed proof of the ZK property.

Completeness. Completeness follows from the completeness of the non-ZK MF-PCS (Theorem 1). The VOLE-based masking does not affect completeness: for instance, in Commit, the masked matrices $D = U^{(1)} - A$ and $E^{(i)} = T^{(i)} - R^{(i)}$ allow both parties to recover the authenticated original matrices via their VOLE shares.

Knowledge soundness. We describe how the extractor Ext extracts the valid witness by emulating the random oracles and $\mathcal{F}_{\text{VOLE}}$. Firstly, since Ext emulates the random oracle for the Merkle tree commitment over the masked matrices D and $\{E^{(i)}\}_{i \in [n+1]}$, so Ext knows their entire contents. Secondly, since Ext also knows the VOLE correlations by emulating the $\mathcal{F}_{\text{VOLE}}$ setup, it can recover $U^{(1)} = D + A$ and $T^{(i)} = E^{(i)} + R^{(i)}$ for each $i \in [n + 1]$. From this point, the extraction proceeds almost identical to the non-ZK case (Theorem 1): the recovered matrices can be uniquely decoded to obtain $\{f^{(i)}\}_{i \in [n]}$ satisfying $\mathcal{R}_{\text{KS}}$.

Zero-knowledge. We prove the ZK property in two parts.

First, we show that com reveals nothing about $f^{(1)}$ by constructing a PPT simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$. We let Sim_1 emulate $\{\mathcal{F}_{\text{VOLE}}^{p_i, r_i}\}_{i \in [n+1]}$ honestly using the randomness ρ , so Sim_1 knows all the VOLE correlations, including $A, \{R^{(i)}\}_{i \in [n+1]}$, and all global/local MAC keys. We let Sim_2 sample uniformly random matrices $\tilde{D} \in \mathbb{F}_{p_1}^{m \times m}$ and $\{\tilde{E}^{(i)} \in \mathbb{F}_{p_i}^{m \times (m' - m)}\}_{i \in [n+1]}$, and compute com as the Merkle tree commitment over $\tilde{D}$ and $\{\tilde{E}^{(i)}\}_{i \in [n+1]}$. In the real protocol, $D = U^{(1)} - A$ and $E^{(i)} = T^{(i)} - R^{(i)}$. Since A and $\{R^{(i)}\}_{i \in [n+1]}$ are uniformly random and independent of $f^{(1)}$ from V's view, the real D and $\{E^{(i)}\}_{i \in [n+1]}$ are identically distributed to $\tilde{D}$ and $\{\tilde{E}^{(i)}\}_{i \in [n+1]}$. Therefore, the simulated com and the real com are indistinguishable.

Second, we show the transcript in VerCom, Eval also reveals nothing about $f^{(1)}$ by constructing a PPT simulator $\text{Sim}' = (\text{Sim}'_1, \text{Sim}'_2)$ for $\mathcal{R}_{\text{KS}}$. The first sub-simulator Sim'_1 is the same as Sim_1 , so Sim'_1 knows all the VOLE correlations. The second sub-simulator Sim'_2 simulates the transcripts of VerCom and Eval as follows.

- **Simulating VerCom:** Sim'_2 reconstructs $\tilde{U}^{(1)} := \tilde{D} + A$ and $\tilde{T}^{(i)} := \tilde{E}^{(i)} + R^{(i)}$, and computes $\tilde{U}^{(i)} := \tilde{U}^{(1)} \bmod p_i$ for $i \in [2, n + 1]$. It then executes the VerCom protocol using these matrices honestly, except for the following situations:
 - For each CheckMod invocation, Sim'_2 invokes the VOLE-ZK simulator of [62] as a sub-routine.
 - For each CheckZero invocation, Sim'_2 computes the correct hash digest $\sigma = H(\mathbf{k})$ using V's local MAC key $\mathbf{k}$, which it knows from the VOLE correlations.
 - For the range proof SNARK for $\mathcal{R}_{\text{Range}}$, Sim'_2 invokes the ZK simulator of the underlying zkSNARK.
- **Simulating Eval:** The simulation of Eval follows the same strategy as VerCom, with the testing phase and evaluation phase simulated analogously. In the evaluation phase, Sim'_2 additionally simulates the final check $\text{CheckZero}((\llbracket \mathbf{w}^{(i)} \rrbracket_{p_i})^T \cdot \mathbf{q}_2^{(i)} - \llbracket y^{(i)} \rrbracket_{p_i})$ using its knowledge of the VOLE correlations.

It remains to argue that the simulated transcript is computationally indistinguishable from the real transcript. The key observation is that all values revealed to V are either (i) masked by uniformly

random VOLE correlations (e.g., the opened columns D^k , which only allows V to obtain $\llbracket U^{(1),k} \rrbracket_{p_1}$ rather than $U^{(1),k}$ in the clear), or (ii) outputs of sub-protocols whose ZK property is guaranteed by their respective simulators. Therefore, we conclude that the simulated transcript is computationally indistinguishable from the real transcript. In conclusion, the construction in Figure 7 satisfies the zero-knowledge property. $\square$

G.4 Proof of Theorem 4

PROOF. Since the construction in Figure 8 is based on the standard SumCheck PIOP, completeness and soundness follow directly from the standard SumCheck PIOP analysis. In the following, we focus on the proof of the ZK property.

First, we construct a PPT simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$ as follows. The first sub-simulator Sim_1 emulates $\mathcal{F}_{\text{VOLE}}^{p,r}$ honestly and generates the VOLE correlations $\llbracket \mathbf{u}_1 \in \mathbb{F}_p^d \rrbracket$ and $\{\llbracket \mathbf{u}_i \in \mathbb{F}_{p^r}^d \rrbracket\}_{i \in [2, \mu]}$. The second sub-simulator Sim_2 simulates the transcript of the i -th round as follows:

- (1) Sim_2 samples uniformly random values $\{\tilde{f}_i(\beta)\}_{\beta \in [0, d] \setminus \{1\}}$.
- (2) Sim_2 executes the Auth procedure honestly using random VOLE correlation $\llbracket \mathbf{u}_i \rrbracket$, so both parties obtain $\llbracket \tilde{f}_i \rrbracket$.
- (3) Sim_2 locally computes $\llbracket \tilde{f}_i(1) \rrbracket := \llbracket H \rrbracket - \llbracket \tilde{f}_i(0) \rrbracket$.
- (4) Upon receiving the challenge α_i from V, Sim_2 updates $\llbracket H \rrbracket := \llbracket \tilde{f}_i(\alpha_i) \rrbracket$, which is computed locally from $\{\llbracket \tilde{f}_i(\beta) \rrbracket\}_{\beta \in [0, d]}$ and α_i via polynomial interpolation.

After μ rounds, both parties hold $\llbracket \tilde{v} \rrbracket := \llbracket \tilde{f}_\mu(\alpha_\mu) \rrbracket$. When V has the oracle access to $\llbracket y \rrbracket$ where $y = f(\alpha_1, \dots, \alpha_\mu)$, Sim_2 needs to pass $\text{CheckZero}(\llbracket \tilde{v} \rrbracket - \llbracket y \rrbracket)$. Since Sim_2 knows all the VOLE correlations by emulating $\mathcal{F}_{\text{VOLE}}^{p,r}$, it can compute V's local MAC key for $\llbracket \tilde{v} \rrbracket - \llbracket y \rrbracket$ and pass the check easily.

Second, we argue that the simulated transcript is indistinguishable from the real transcript. In the real world, V's view in the i -th round consists of $\mathbf{d}_i = \mathbf{f}_i - \mathbf{u}_i$, where $\mathbf{f}_i = (f_i(\beta))_{\beta \in [0, d] \setminus \{1\}}$ is the real round polynomial evaluation vector. In the ideal world, V's view is $\tilde{\mathbf{d}}_i = \tilde{\mathbf{f}}_i - \mathbf{u}_i$, where $\tilde{\mathbf{f}}_i$ is uniformly random. Since $\mathbf{u}_i$ is uniformly random, $\mathbf{d}_i$ and $\tilde{\mathbf{d}}_i$ are perfectly indistinguishable. Furthermore, by the hiding property of VOLE correlations, V cannot learn anything about the authenticated values. The final CheckZero step is also correctly simulated since Sim_2 computes the exact hash digest that V expects. Therefore, we conclude that the real transcript is indistinguishable from the simulated transcript.

In conclusion, the construction in Figure 8 satisfies the zero-knowledge property. $\square$

G.5 Proof of Theorem 5

PROOF. Construction. We construct a zkSNARK $\Pi = (\Pi.\text{Setup}, \Pi.\text{Prove}, \Pi.\text{Verify})$ for $\mathcal{R}_{\text{EvalOp}}$ using PCS, MF-PCS and PIOPs.

- $\Pi.\text{Setup}$: Invoke the setup algorithms of PCS and MF-PCS, and generate sufficient random VOLE correlations.
- $\Pi.\text{Prove}$ and $\Pi.\text{Verify}$: We follow the “PIOP+PCS” paradigm. As defined in Definition 5, EvalOp comprises additions, multiplications, permutations, (I)NTT operations, and cross-field modulo operations. The proving procedure works as follows:
 - (1) P first commits to the witness $\mathbf{w}$ using PCS.

- (2) For additions, multiplications, permutations, and (I)NTT operations over a fixed field, P commits to the computed results using PCS, then proves the correctness of these results using the corresponding SumCheck-based PIOPs (including our NTT PIOP from Section 5).

We note that, the PIOPs may involve polynomials of degree higher than one, while the (ZK) PCS only commits to multilinear polynomials. An additional bridging step is needed when the PIOP's final evaluation query falls on a higher-degree polynomial. For example, to prove a multiplication relation $z = x \odot y$, we represent x, y, z as multilinear polynomials f_x, f_y, f_z and invoke a ZeroCheck PIOP on the degree-2 polynomial $g := f_z - f_x \cdot f_y$. At the end of the ZeroCheck, V needs the evaluation $g(\mathbf{r})$ at a random point $\mathbf{r}$. However, the ZK PCS can only provide V with authenticated evaluations $\llbracket f_x(\mathbf{r}) \rrbracket, \llbracket f_y(\mathbf{r}) \rrbracket, \llbracket f_z(\mathbf{r}) \rrbracket$. To bridge this gap, P computes $g(\mathbf{r}) = f_z(\mathbf{r}) - f_x(\mathbf{r}) \cdot f_y(\mathbf{r})$ and invokes Auth to let V obtain $\llbracket g(\mathbf{r}) \rrbracket$. Then, P uses the CheckMult procedure together with the linear homomorphism of VOLE to convince V that $\llbracket g(\mathbf{r}) \rrbracket$ is correctly computed. The same technique applies to other PIOPs whose final evaluation involves nonlinear combinations of committed multilinear polynomials.

- (3) For cross-field modulo operations, P commits to the cross-field values using MF-PCS.Commit. Since MF-PCS inherently provides oracle access to the committed polynomial over each target field $\mathbb{F}_{p_i}$, these commitments can be directly referenced by subsequent PIOPs without recommitting. When V requires polynomial evaluations, P can simply run MF-PCS.VerCom and MF-PCS.Eval with V.
- (4) After all basic operations of EvalOp have been proven, P and V hold $\llbracket \mathbf{o} \rrbracket$. Finally, P opens $\llbracket \mathbf{o} \rrbracket$ to V.

Completeness. Completeness follows directly from the completeness of each building block.

Knowledge soundness. The knowledge soundness of Π follows from the standard “PIOP+PCS” compilation framework [13, 21]. More precisely, suppose there exists a PPT adversary A can make the verifier accept with non-negligible probability. By the knowledge soundness of PCS and MF-PCS, there exist extractors that recover the committed polynomials from A's commitments. By the (knowledge) soundness of each PIOP instance, the extracted polynomials must satisfy the corresponding relations with overwhelming probability. Since EvalOp is decomposed into a sequence of basic operations and each operation is individually proven, the composition of all extracted relations implies $\mathbf{o} = \text{EvalOp}(\mathbf{w})$. The overall soundness error is bounded by the union bound over all PIOP and PCS instances, which remains negligible.

Zero-knowledge. The ZK property of Π follows from the ZK properties of its building blocks. Since all intermediate values are hidden by VOLE-based masking, and the only value revealed to V is the output $\mathbf{o}$ (which is part of the public statement), the overall protocol reveals nothing beyond the statement $(\text{EvalOp}, \mathbf{o})$. The simulator for Π can be constructed by composing the simulators of ZK (MF-)PCS (Theorem 3), ZK SumCheck PIOP (Theorem 4) and the VOLE-ZK protocols for multiplications [60] and cross-field modulo operations [62]. $\square$

H More Details on Complexity Analysis

H.1 Complexity Analysis on ZK MF-PCS

Recall that $M = m^2$, $R = \sum_{i \in [n]} r_i$. We let $\tilde{n}(m)$, $\tilde{t}_P(m)$, $\tilde{t}_V(m)$, $\tilde{p}(m)$ denote the number of required random VOLE correlations, the prover time, the verifier time and the proof size of each invocation of CheckMod on length- m vectors over small fields; when the auxiliary field is involved, we will take the additional $O(\log M)$ overhead into consideration.

In our experiments, we instantiate the linear code with RS code with code rate 0.5, so we have $m' = 2m$ and $t_C(m) = O(m \log m)$. We instantiate CheckMod with the fixed protocol in ZHE [62], whose dominant cost is an execution of a Boolean circuit of $O(\beta)$ multiplication gates, where β is the bit length of the underlying field. When the field parameters are fixed, we omit the constant factor β and conclude that $\tilde{n}(m)$, $\tilde{t}_P(m)$, $\tilde{t}_V(m)$, $\tilde{p}(m)$ are all $O_\lambda(m)$. Given these instantiations, we analyze the complexity as follows.

Offline phase. As shown in Figure 7, the parties need $O_\lambda(nM)$ (resp. $O_\lambda(M + R\sqrt{M})$) random VOLE correlations over the small fields (resp. the auxiliary field). In addition, the parties also need $O_\lambda(R\tilde{n}(m))$ correlations over both small fields and the auxiliary field for the CheckMod invocations in online phase.

In our experiments, we assume there is a Trusted Third Party (TTP) who is responsible for generating these offline materials, so both the prover and the verifier need to take $O_\lambda((n + \log M)M + R\sqrt{M} \log M)$ time to read and store these materials. Note that, the offline phase can also be instantiated with the secure two-party (2PC) protocols: To generate random VOLEs, we can use the protocol in [57], whose time complexity for both parties is linear in the input size. As for CheckMod, we can use the “cut-and-choose” protocol in [62] to generate the required correlations, whose time complexity for both parties is also linear in the input size. As a result, the time complexity for both parties is $O_\lambda((n + \log M)M + R\sqrt{M} \log M)$.

Online phase. For Commit protocol, the prover time and commitment size remain unchanged from the non-ZK version, since the only additional operation is masking the committed matrices with random VOLEs. For VerCom and Eval protocols, the extra work consists mainly of Auth, CheckZero, and CheckMod on $O(n\lambda)$ (resp. $O(R + \lambda)$) vectors of length $\sqrt{M}$ over small fields (resp. the auxiliary field). Consequently, when the field parameters are fixed, the online prover time is $O_\lambda(R\sqrt{M} \log M + t_P(M))$, the online verifier time is $O_\lambda(R\sqrt{M}(\log M)^2 + t_V(M))$ and the proof size is $O_\lambda(R\sqrt{M} \log M + p(M))$, where $t_P(M)$, $t_V(M)$, $p(M)$ denote the prover time, verifier time and proof size of the zk-SNARK for $\mathcal{R}_{\text{Range}}$, respectively. Since $t_V(M)$, $p(M)$ are sublinear in M by the definition of zk-SNARK, our online verifier time and proof size are sublinear in M .

H.2 Complexity Analysis on ZK SumCheck PIOP

In our experiments, the polynomial degree d and the extension degree r are set as small constants, i.e., $d = O(1)$, $r = O(1)$. Given these, we analysis the complexity as follows.

Offline phase. As stated in Figure 8, the parties need d (resp. $d(\mu - 1)$) random VOLE correlations over $\mathbb{F}_p$ (resp. $\mathbb{F}_{p^r}$). Therefore, the number of required random VOLE correlations is $O(\log N)$.

When these correlations are generated by a TTP, the prover and the verifier need to take $O(\log N)$ time to read and store these correlations. When these correlations are generated via the 2PC protocol in [57], both the prover and the verifier’s time are $O(\log N)$.

Online phase. In the non-ZK SumCheck protocol, P sends f_i directly to V in the i -th round; while in the ZK SumCheck protocol, P invokes Auth with V to obtain $\llbracket f_i \rrbracket$. This replacement only incurs additional $O(\log N)$ operation for both parties, and no extra communication is introduced. Since the rest of the ZK protocol is similar to the non-ZK protocol, we conclude that the online prover time is $O(N)$, both the online verifier time and the proof size are $O(\log N)$.

I Discussions

Here we provide discussions on bootstrapping and public verifiability.

Bootstrapping. The experiments conducted in Section 8 do not involve an important procedure named bootstrapping. Bootstrapping achieves unbounded multiplications by refreshing a low-level ciphertext and lifting it to a higher level to enable more multiplications. As discussed in ZHE [62], CKKS’s bootstrapping is indeed an admissible HE evaluation. Therefore, according to Theorem 5, PRISM can be used to prove the correct execution of bootstrapping.

Public verifiability. The non-ZK version of PRISM is publicly verifiable (i.e., any party can act as a verifier), while the ZK version is a designated-verifier zkSNARK due to the use of VOLE correlations. Traditional approaches for adding zero-knowledge to proof systems, such as introducing random masking polynomials, can preserve public verifiability and have been successfully applied to SumCheck-based PIOPs [58] and Brakedown-style PCSs [2, 59]. However, it remains an open question how to apply these traditional techniques to make MF-PCS zero-knowledge, since MF-PCS involves cross-field consistency checks that do not have a straightforward analogue in the standard masking-based framework. We leave the construction of a publicly verifiable ZK MF-PCS as an interesting direction for future work.

J More Details about Range Proof Instantiation

We show how to employ the table-decomposition technique [48] to instantiate the range proof. For example, in order to prove $0 \leq v < p$, where p is a public 48-bit prime, both parties compute the public 16-bit limbs $\{\rho_i\}_{i \in [3]}$ of p such that $p = \sum_{i \in [3]} \rho_i 2^{16(i-1)}$. The prover similarly decomposes v into 16-bit limbs $\{v_i\}_{i \in [3]}$, and proves $v = \sum_{i \in [3]} v_i 2^{16(i-1)}$ to the verifier. Proving $v < p$ is then equivalent to proving that (v_3, v_2, v_1) is lexicographically smaller than (ρ_3, ρ_2, ρ_1) ; namely, $(v_3 < \rho_3) \vee (v_3 = \rho_3 \wedge v_2 < \rho_2) \vee (v_3 = \rho_3 \wedge v_2 = \rho_2 \wedge v_1 < \rho_1)$.

To prove this relation, for each $i \in [3]$, both parties construct a public table $T_i = \{(x, \text{cmp}(x, \rho_i)) : x \in [0, 2^{16})\}$, where $\text{cmp}(x, \rho_i)$ is LT, EQ, or GT according to whether $x < \rho_i$, $x = \rho_i$, or $x > \rho_i$, respectively. Since ρ_i is public, T_i can be constructed publicly. By proving a lookup of (v_i, c_i) into T_i , the prover simultaneously establishes $0 \leq v_i < 2^{16}$ and proves that $c_i \in \{\text{LT}, \text{EQ}, \text{GT}\}$ correctly represents the comparison between v_i and ρ_i . The above lexicographic relation is therefore equivalent to $(c_3 = \text{LT}) \vee (c_3 = \text{EQ} \wedge c_2 = \text{LT}) \vee (c_3 = \text{EQ} \wedge c_2 = \text{EQ} \wedge c_1 = \text{LT})$.

To prove the above relation, we encode LT, EQ, GT as 0, 1, 2, respectively, and pack the three comparison results into $s = c_1 +$

$3c_2 + 9c_3$. The three cases above correspond to $s \in \{0, \dots, 8\}$, $s \in \{9, 10, 11\}$, and $s = 12$, respectively. Hence, the required relation holds if and only if $s \in \{0, \dots, 12\}$. We enforce this condition with one lookup into a public table T_{lex} of size 32, where $T_{\text{lex}}(x) = x$ for $0 \leq x \leq 12$ and $T_{\text{lex}}(x) = x + 32$ for $13 \leq x < 32$. The prover proves a lookup of (s, s) into T_{lex} , which succeeds exactly when $s \in \{0, \dots, 12\}$. Together with the limb-decomposition constraint, this establishes $0 \leq v < p$ and completes the range proof.

K More Evaluations

In Section 8.1, we evaluated our MF-PCS and compared PRISM with related works in Section 8.3. Here we report the performance of other basic building blocks (Appendix K.1) and provide additional evaluation results of PRISM (Appendix K.2).

K.1 Performance of Basic Building Blocks

Performance of (zk) SumCheck PIOP. We benchmark the Sum-Check PIOP on the task of proving $H = \sum_{b \in B_\mu} (\prod_{i \in [d]} f_i(b))$ for d multilinear polynomials $f_i \in \mathcal{F}_{\mu, p}^{(\leq 1)}$. For a sub-field $\mathbb{F}_p$ and its extension field $\mathbb{F}_{p^r}$, we let p be a 48-bit prime and set $r = 3$. We fix $d = 2$, vary the polynomial size N from 2^{10} to 2^{20} and report results for both the non-ZK and ZK versions. The ZK offline phase generates the required VOLE correlations; its cost ranges from 13.1 ms ($N = 2^{10}$) to 26.6 ms ($N = 2^{20}$), which is lightweight compared to the online phase. The prover time, verifier time, and proof size are shown in Figure 13.

The ZK overhead on the prover is negligible (less than 5%), since the ZK SumCheck protocol (Figure 8) only introduces lightweight VOLE-based masking operations. Figure 13(b) shows that the verifier time grows logarithmically with N , confirming the $O(\log N)$ complexity. Because the baseline verifier computation in SumCheck is already very fast, the additional VOLE-based operations in the ZK version lead to a more noticeable relative overhead (1.3×–1.4×). As shown in Figure 13(c), the proof size is sublinear in N and identical for both versions, since the ZK protocol does not incur additional communication.

Performance of (zk) Brakedown PCS. We evaluate the batched version of Brakedown PCS, where $2^8 = 256$ small multilinear polynomials are packed into a single large polynomial and committed together. We conduct the experiments over a subfield $\mathbb{F}_p$ and its extension field $\mathbb{F}_{p^r}$, where p is a 48-bit prime and $r = 3$. We test three polynomial sizes $N \in \{2^{12}, 2^{13}, 2^{14}\}$, matching the parameter choices in the application scenarios of Section 8.3. The amortized results per small polynomial are reported in Table 6.

As shown in Table 6, the ZK offline phase is lightweight (at most 19.4 ms), as it only generates VOLE correlations; the non-ZK version requires no offline phase. The ZK overhead on the prover side ranges from 10% to 24%, due to the additional VOLE-based masking and the CheckZero invocations. Unlike the MF-PCS, the proof size is nearly identical for both versions of Brakedown PCS, since the ZK version of Brakedown PCS only involves the CheckZero invocations, while the ZK version of MF-PCS additionally involves the CheckMod invocations which are more expensive. Note that, the CheckZero procedures only incur an additional 32 bytes of communication, which is negligible compared to the total proof size.

K.2 More Evaluations of PRISM

Table 7 presents the full evaluation results of both the non-ZK and ZK versions of PRISM across all the scenarios (i.e., NN, CM, MVM, LR and CA) described in Section 8.3.

The non-ZK version achieves a fast offline phase (at most 0.14 s for the LR scenario), since it only involves the lightweight offline phase of the NTT PIOP. The ZK version incurs a higher offline cost due to the generation of VOLE correlations, ranging from 1.08 s to 101.63 s. Nevertheless, the ZK overhead on the online prover time remains small (roughly 5%–7% across all scenarios), since the VOLE-based ZKP is prover-efficient. The verifier time increases by roughly 3×–3.5× and the proof size by roughly 1.7×–2.1× in the ZK version, mainly due to the ZK overhead of MF-PCS, as discussed in Section 8.1.

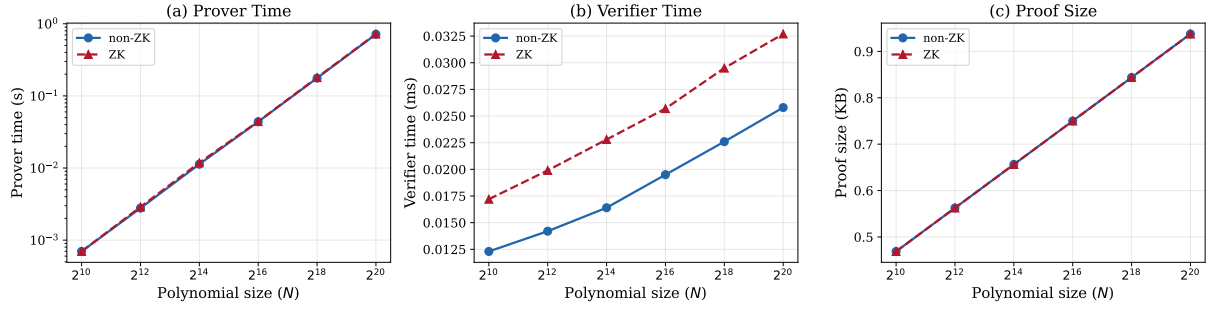


Figure 13: Performance of (zk) SumCheck PIOP for proving $H = \sum_{b \in B_\mu} f(b) \cdot g(b)$ with varying polynomial size N .

Table 6: Amortized performance of (zk) Brakedown PCS per small polynomial.

N	Offline time (ms)		Commit time (ms)		Eval time (ms)		Verifier time (ms)		Proof size (KB)	
	non-ZK	ZK	non-ZK	ZK	non-ZK	ZK	non-ZK	ZK	non-ZK	ZK
2^{12}	–	4.70	1.28	1.45	0.91	1.09	0.10	0.28	1.99	1.99
2^{13}	–	9.80	2.74	3.07	1.76	2.01	0.14	0.32	2.58	2.58
2^{14}	–	19.4	5.83	6.49	3.53	3.84	0.23	0.45	3.17	3.17

Table 7: More evaluation results of both non-ZK version and ZK version of PRISM.

Scenario	Scheme	N	L	HE time (ms)	Offline time (s)		Prover time (s)		Verifier time (s)		Proof size (MB)	
					non-ZK	ZK	non-ZK	ZK	non-ZK	ZK	non-ZK	ZK
NN	CKKS	2^{12}	1	4.58	0.01	1.08	5.50	5.85	0.09	0.31	1.87	3.12
	BGV	2^{12}	1	6.02	0.01	2.16	9.64	10.28	0.17	0.57	3.30	5.98
CM	CKKS	2^{13}	2	3.45	0.03	1.55	3.22	3.39	0.04	0.13	0.68	1.14
	BGV	2^{13}	2	4.17	0.03	1.92	4.59	4.84	0.06	0.19	0.96	1.74
MVM	CKKS	2^{13}	2	36.88	0.03	19.43	44.14	46.68	0.56	1.78	9.24	16.16
	BGV	2^{13}	2	43.54	0.03	25.31	64.60	68.22	0.83	2.74	13.44	25.05
LR	CKKS	2^{14}	6	179.17	0.14	101.63	181.87	191.05	2.03	6.07	22.28	46.16
CA	CKKS	2^{14}	7	554.65	0.16	326.99	547.10	572.92	6.33	18.69	70.02	142.20