

# Practical Zero-Knowledge PIOP for Public Key and Ciphertext Generation in (Multi-Group) Homomorphic Encryption

Intak Hwang  
Seoul National University

Hyeonbum Lee  
Hanyang University

Jinyeong Seo  
Seoul National University

Yongsoo Song  
Seoul National University

**Abstract**—Homomorphic encryption (HE) is a foundational technology in privacy-enhancing cryptography, enabling non-interactive computation over encrypted data. Recently, generalized HE primitives designed for multi-party applications, such as multi-group HE (MGHE), have gained significant research interest. While constructing secure multi-party protocols from (MG)HE in the semi-honest model is straightforward, zero-knowledge techniques are essential for ensuring security against malicious adversaries.

In this work, we design practical proof systems for MGHE to guarantee the well-formedness of public keys and ciphertexts. Specifically, we develop and optimize a polynomial interactive oracle proof (PIOP) for MGHE, which can be compiled into zk-SNARKs using a polynomial commitment scheme (PCS).

We compile our PIOP using a lattice-based PCS, and our implementation achieves a 5.5x reduction in proof size, a 70x speed-up in proof generation, and a 343x improvement in verification time compared to the previous state-of-the-art construction, PELTA (ACM CCS 2023). Additionally, our PIOPs are modular, enabling the use of alternative PCSs to optimize other aspects, such as further reducing proof sizes.

**Keywords**—homomorphic encryption, zero-knowledge proof, multiparty computation, malicious security

## 1. Introduction

In recent years, Homomorphic Encryption (HE) has been widely adopted in various privacy-preserving protocols, such as private information retrieval [1, 2], private set intersection [3, 4], oblivious message retrieval [5, 6], and secure inference [7, 8], demonstrating its versatility and practicality.

Despite these advantages, there are issues that cannot be resolved using HE alone, of which we highlight two primary limitations. First, in conventional HE, homomorphic operations are only possible when ciphertexts are encrypted under the same key, making the key owner as a single point of failure.

To overcome this limitation, various solutions have been proposed for trust distribution across multiple entities, and currently multi-party HE (MPHE) [9] or multi-key HE (MKHE) [10] are the most promising solutions. In MPHE, all participants are involved in a key-generation protocol to

obtain secret key shares and build a joint public key for HE. In the MKHE setting, each party separately generates its own key pair so the main challenge is in the design of homomorphic operation algorithms supporting computation between ciphertexts encrypted under different keys. Recently, Kwak et al. [11] developed a new notion of multi-group HE (MGHE) that combines the best of both worlds, making homomorphic computation between multiple entities more practical.

On the other hand, integrity is another issue that must be addressed when using HE. Existing literature often assumes a semi-honest model when building protocols with HE. However, while the security definition of HE ensures data privacy, it does not guarantee integrity against attacks from malicious adversaries. For example, recent studies presented potential vulnerabilities of HE-based protocols under malicious settings in the context of IND – CPA<sup>D</sup> attacks [12, 13, 14]. In contrast to the extensive research focused on improving the performance of HE, efforts to enhance the security of HE-based protocols have been quite limited and remain largely theoretical.

Achieving malicious security in HE-based protocols has been discussed in various directions [15, 16, 17, 18, 19], but the most viable solution is attaching non-interactive zero-knowledge proofs (NIZK) to verify the correct execution of the protocol, thus preventing malicious behaviors. This naturally leads to the demand for efficient NIZKs, or more specifically zk-SNARKs (succinct non-interactive arguments of knowledge), for HE ciphertexts and public keys. As a result, a sequence of studies [20, 21, 22, 23] focused on constructing efficient proof systems for HE ciphertexts and public keys. However, these solutions are still far from practical deployment due to significant overheads in proof generation time or proof size. Additionally, they do not employ recent advances in SNARK constructions, such as the polynomial interactive oracle proof (PIOP) framework.

### 1.1. Our Contributions

In this work, we construct efficient proof systems that ensures public key and ciphertext integrity in HE systems. Specifically, our proof system is based on the PIOP framework, reflecting recent advances in SNARKs. In the PIOP framework [24, 25], given an NP-relation to be proven, we first design a specially formed interactive proof system,

called PIOP. Then, this can be compiled into a SNARK using a polynomial commitment scheme and applying the Fiat-Shamir transform. This provides modularity in designing SNARKs, as it suffices to construct an efficient PIOP for the given relation, which is automatically transformed into a SNARK through the PIOP compilation. In this context, we primarily focus on designing efficient PIOP for MGHE ciphertexts and public keys.

**PIOP over Polynomial Ring.** We first design efficient PIOPs for polynomials, including proofs for arithmetic relations and bounds on the coefficients, as most practical HE schemes [26, 27] are constructed over polynomial rings. To achieve this, we utilize the number-theoretic transform (NTT) to convert constraints over polynomials into constraints over vectors, since existing PIOPs [28, 24] are optimized for handling vectors in finite fields. These include PIOPs such as the row check PIOP for verifying Hadamard products between vectors and the linear check PIOP for verifying linear relations between vectors. However, these PIOPs are insufficient for our cases, so we design two new PIOPs: the *generalized row check* PIOP and the *norm check* PIOP, which handle more general constraints over vectors.

**PIOP for MGHE.** Based on our PIOPs for polynomial rings, we design practical PIOPs that specifically prove the well-formedness of public keys and ciphertexts of an MGHE scheme [11]. However, there still remains a gap between the input space for PIOP and the ciphertext space of HE due to differences in coefficient moduli. For PIOP, the coefficient modulus must be a large prime to ensure negligible soundness error, whereas HE ciphertext modulus is typically a product of small primes for efficient polynomial arithmetic. We bridge this gap by introducing a modulus-switching technique, temporarily converting the modulus of HE ciphertexts into a large prime during proof generation. We remark that the modulus-switching technique was originally introduced to be used in homomorphic multiplication but we reuse the same technique for efficient proof generation.

For proving constraints related to HE ciphertexts and public keys, we frequently need to verify the NTT operation, which can be represented as a matrix-vector multiplication using a Vandermonde matrix. We can prove this using the linear check PIOP. However, during the linear check PIOP, a random vector needs to be multiplied by the transpose of a Vandermonde matrix, whose complexity is quadratic with vector dimension. To resolve this, we utilize algebraic properties of Vandermonde matrices and employ inverse NTT operations to achieve quasi-linear complexity.

**Concrete Efficiency.** For a concrete instantiation, we compile our PIOP for MGHE using a polynomial commitment scheme (PCS). Among various candidates for PCS, we specifically use the lattice-based construction by Hwang et al. [29] as it provides fast proof generation and post-quantum security. One downside is its proof size, which scales with the square root of the input size; however, it still results in a smaller proof size compared to HE public keys.

For benchmarking, we measure proof size and runtimes for both the prover and verifier across various types of

public keys. For encryption keys, our PIOP achieves a 5.5x smaller proof size, 70x faster proof generation, and 343x faster verification compared to the previous state-of-the-art construction by Chatel et al. [23]. We expect the gap in proof size to widen further if larger keys, such as relinearization and automorphism keys, are included, as our proof size grows at a square-root rate, while the previous work scales linearly. Thanks to the modularity of the PIOP, we note that the proof size of our PIOP could be further reduced, at the cost of slower proof generation, if compiled with other PCS [30, 31, 32, 33], which offer polylogarithmic or constant proof sizes.

## 1.2. Applications

**Maliciously Secure MPC.** The most straightforward application of our PIOP is achieving malicious security in MGHE-based MPC protocols. In [16, 15], round-efficient general MPC protocols are proposed based on the functionalities of MKHE and MPHE, which can be replaced by MGHE. Initially, these protocols are designed to be secure against semi-malicious adversaries but can be compiled to a fully malicious setting by incorporating NIZKs for ciphertexts and public keys. Since our PIOP naturally produces zk-SNARKs for MGHE ciphertexts and public keys, it can be utilized to enhance the security of MGHE-based MPC protocols.

**Malicious Circuit Privacy.** HE is often utilized in a client-server scenario where the server also has private input for homomorphic evaluation. This setup is particularly useful in asymmetric settings, where the server’s input is much larger than the client’s input, such as in private large language models [34, 35] or private set intersection [3, 4]. However, this requires that the protocol does not leak information about the server’s input or, more generally, about the circuit, which is referred to as circuit privacy. Circuit privacy can be extended to an MKHE setting [19], which is well-suited for the security model in collaborative inference. Practical solutions for circuit privacy are typically based on the noise-flooding technique [3], which assumes a semi-honest setting, where the client behaves honestly. This can be enhanced to a fully malicious setting by incorporating NIZKs for HE ciphertexts and public keys, as noted in [18]. Thus, our PIOP can be utilized to achieve circuit privacy against malicious clients.

**Setup for SPDZ.** The SPDZ [36] protocol is a secret-sharing-based MPC protocol that achieves security against malicious adversaries. During the offline phase of SPDZ, specially structured randomness, called authenticated triples, are generated through MPHE functionality. While it provides efficient proof systems for verifying MPHE ciphertexts, a proof system for joint public keys in MPHE is not precisely described. Recently, Rotaru et al. [37] proposed an MPC-based solution for constructing these public keys securely, which requires about 2 hours in the two-party case. We conceive that our PIOP for MGHE public keys can be

another solution to this problem, as it can generate all the required proofs for joint key generation in 13 minutes.

### 1.3. Related Work

**NIZK for HE.** Previously, several works in the literature have constructed NIZKs for HE. Del Pino et al. [20] utilize the inner product argument from Bulletproof [38] to represent constraints for HE ciphertexts. However, they do not provide concrete performance metrics. Boschini et al. [21] construct a proof system based on the R1CS representation in Aurora [28], which is based on PIOP. This approach achieves a short proof size for HE ciphertexts due to its polylogarithmic complexity, but it results in slow proof generation due to a lack of PIOP-level optimization. The most relevant work to ours is the proof system by Chatel et al. [23], which proves constraints for MPHE [9] ciphertexts and public keys. To achieve fast proof generation, they use the LANES [39, 40, 41] framework, which provides fast proof generation based on lattice-based cryptography but results in a large proof size that scales linearly with the input size. Compared to other previous work, it presents proof systems for evaluation keys, which have a more complex structure than ciphertexts. However, since the LANES framework is optimized for a small coefficient modulus, originally targeting lattice-based signatures, it requires dozens of repetitions to generate proofs for HE ciphertexts, as the coefficient modulus of HE is a product of dozens of small primes.

**Verifiable Computation.** In addition to ensuring the integrity of HE ciphertexts and public keys, another line of research [42, 43, 22] focuses on validating the integrity of homomorphic computations. While this is orthogonal to our work, our methodology is technically related, as it employs SNARKs to prove the validity of computations, which are specifically optimized for sublinear verification complexity. Hence, we believe that our PIOP for polynomial rings can be utilized to construct efficient SNARKs for HE operations within the PIOP framework.

## 2. Background

### 2.1. Notation

For a positive integer  $q$ , we use  $\mathbb{Z} \cap (-q/2, q/2]$  as a representative set of  $\mathbb{Z}_q$ , and denote by  $[a]_q$  the reduction of  $a$  modulo  $q$ . Vectors over  $\mathbb{Z}$  or  $\mathbb{Z}_q$  are denoted with regular lowercase letters and arrows, such as  $\vec{v}$ , and matrices over  $\mathbb{Z}$  or  $\mathbb{Z}_q$  are represented by regular uppercase letters. We regard all vectors as column vectors, and we use the symbol  $\|$  for the concatenation of two vectors.

Let  $d$  be a power of two. We denote by  $R = \mathbb{Z}[X]/(X^N + 1)$  the ring of integers of the  $2N$ -th cyclotomic field and  $R_q = \mathbb{Z}_q[X]/(X^N + 1)$  the residue ring of  $R$  modulo  $q$ . For polynomials, we use bold lowercase letters to

denote them e.g.,  $\mathbf{f}$ . For a vector  $\vec{v} = (v_0, \dots, v_{n-1}) \in \mathbb{Z}^n$ , the  $\delta^p$  ( $p \geq 1$ ) and  $\delta^\infty$  norms are defined as follows.

$$\|v\|_p := \sqrt[p]{\sum_{i=0}^{n-1} |v_i|^p} \quad \text{and} \quad \|v\|_\infty := \max_{0 \leq i < n} |v_i|$$

The Hadamard product is denoted by  $\circ$ . For a polynomial  $\mathbf{f}$  or a vector of polynomials  $\vec{\mathbf{f}}$ ,  $\|\mathbf{f}\|_p$  and  $\|\vec{\mathbf{f}}\|_p$  are calculated by regarding them as coefficient vectors. For a matrix  $A \in \mathbb{R}^{n \times n}$ , we denote the matrix norm of  $A$  by  $\|A\|_2 := \max_{0 \neq \vec{x} \in \mathbb{R}^n} \frac{\|A\vec{x}\|_2}{\|\vec{x}\|_2}$ .

### 2.2. Probability Distributions

We denote sampling  $x$  from the distribution  $\mathcal{D}$  by  $x \leftarrow \mathcal{D}$ . For distributions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  over a countable set  $S$  (e.g.  $\mathbb{Z}^n$ ), the statistical distance of  $\mathcal{D}_1$  and  $\mathcal{D}_2$  is defined as  $\frac{1}{2} \cdot \sum_{x \in S} |\mathcal{D}_1(x) - \mathcal{D}_2(x)| \in [0, 1]$ . We denote the uniform distribution over  $S$  by  $\mathcal{U}(S)$  when  $S$  is finite.

We define the  $n$ -dimensional spherical Gaussian function  $\rho : \mathbb{R}^n \rightarrow (0, 1]$  as  $\rho(\vec{x}) := \exp(-\pi \cdot \vec{x}^\top \vec{x})$ . In general, for a positive definite matrix  $\Sigma \in \mathbb{R}^{n \times n}$ , we define the elliptical Gaussian function  $\rho_{\sqrt{\Sigma}} : \mathbb{R}^n \rightarrow (0, 1]$  as  $\rho_{\sqrt{\Sigma}}(\vec{x}) := \exp(-\pi \cdot \vec{x}^\top \Sigma^{-1} \vec{x})$ . Let  $\Lambda \subseteq \mathbb{R}^n$  be a lattice and  $\vec{c} \in \mathbb{R}^n$ . The discrete Gaussian distribution  $\mathcal{D}_{\vec{c} + \Lambda, \sqrt{\Sigma}}$  is defined as a distribution over the coset  $\vec{c} + \Lambda$ , whose probability mass function is  $\mathcal{D}_{\vec{c} + \Lambda, \sqrt{\Sigma}}(\vec{x}) = \rho_{\sqrt{\Sigma}}(\vec{x}) / \rho_{\sqrt{\Sigma}}(\vec{c} + \Lambda)$  for  $\vec{x} \in \vec{c} + \Lambda$  where  $\rho_{\sqrt{\Sigma}}(\vec{c} + \Lambda) := \sum_{\vec{v} \in \vec{c} + \Lambda} \rho_{\sqrt{\Sigma}}(\vec{v}) < \infty$ . When  $\Sigma = \sigma^2 \cdot I_n$  for  $\sigma > 0$  and the  $n$ -dimensional identity matrix  $I_n$ , we substitute  $\sqrt{\Sigma}$  by  $\sigma$  in the subscript and refer to  $\sigma$  as the width parameter. For a polynomial  $\mathbf{f}$  with  $\deg \mathbf{f} < n$ , we denote by  $\mathbf{f} \leftarrow \mathcal{D}_{\vec{c} + \Lambda, \sqrt{\Sigma}}$  if we sample its coefficient vector from  $\mathcal{D}_{\vec{c} + \Lambda, \sqrt{\Sigma}}$ .

### 2.3. Polynomial Commitment Scheme

A polynomial commitment scheme (PCS) is a class of commitment scheme that takes polynomials as messages and allows the evaluation of committed polynomials. Below, we define a polynomial commitment scheme for univariate polynomials, adapted from [24].

**Definition 1** (Polynomial Commitment). *A polynomial commitment PC consists of the following PPT algorithms.*

- **PC.Setup**( $1^\lambda, D$ )  $\rightarrow$  ck: Given a security parameter  $\lambda$  and a global polynomial degree upper bound  $D$ , it generates a commitment key ck.
- **PC.Com**(ck,  $d, \mathbf{f}$ )  $\rightarrow$  (c,  $\delta$ ): Given a polynomial  $\mathbf{f} \in \mathbb{Z}_p[X]$  with degree  $< d$ , it generates a commitment  $c$  and an opening hint  $\delta$ .
- **PC.Open**(ck,  $c, d, \mathbf{f}, \delta$ )  $\rightarrow$  b: Given a commitment  $c$ , a polynomial  $\mathbf{f}$  with degree  $< d$ , and an opening hint  $\delta$ , it outputs 0 or 1.
- **PC.Eval**(ck,  $x, d, \mathbf{f}, \delta$ )  $\rightarrow$  (y,  $\rho$ ): Given an evaluation point  $x \in \mathbb{Z}_p$  and an opening hint  $\delta$ , it returns an evaluation result  $y$ , and an evaluation proof  $\rho$ .

- **PC.Check**(ck, c, d, x, y, ρ) → b: Given a commitment c, a degree upper bound d, an evaluation point x, an evaluation result y, and an evaluation proof ρ, it outputs 0 or 1.

PC is called a polynomial commitment scheme if it satisfies the following properties.

- **Correctness**: For every polynomial  $f \in \mathbb{Z}_p[X]$  with a degree upper bound  $d \leq D$  and every point  $x \in \mathbb{Z}_p$ , the following holds.

$$\Pr \left[ \begin{array}{l} \text{PC.Open}(ck, c, d, f, \delta) = 1 \wedge \\ \text{PC.Check}(ck, c, d, x, f(x), \rho) = 1 \end{array} \middle| \begin{array}{l} ck \leftarrow \text{PC.Setup}(1^\lambda, D) \\ (c, \delta) \leftarrow \text{PC.Com}(ck, d, f) \\ (y, \rho) \leftarrow \text{PC.Eval}(x, \delta) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

- **Extractability**: For every PPT adversary  $A$ , there exists a PPT extractor  $E$  such that for all randomness  $r$ , the following holds.

$$\Pr \left[ \begin{array}{l} \text{PC.Check}(ck, c, d, x, y, \rho) = 1 \wedge \\ (\text{PC.Open}(ck, c, d, f, \delta) = 0 \vee \\ y \neq f(x)) \end{array} \middle| \begin{array}{l} ck \leftarrow \text{PC.Setup}(1^\lambda, D) \\ (c, d, x, y, \rho) \leftarrow A(ck, r) \\ (f, \delta) \leftarrow E(ck, r) \end{array} \right] \leq \text{negl}(\lambda)$$

- **Binding**: For every PPT adversary  $A$ , the following holds.

$$\Pr \left[ \begin{array}{l} \text{PC.Open}(ck, c, d, f, \delta) = 1 \wedge \\ \text{PC.Open}(ck, c, d, f', \delta') = 1 \wedge \\ f \neq f' \end{array} \middle| \begin{array}{l} ck \leftarrow \text{PC.Setup}(1^\lambda, D) \\ (c, d, f, f', \delta, \delta') \leftarrow A(ck) \end{array} \right] \leq \text{negl}(\lambda)$$

PC is called hiding if the following property holds.

- **Hiding**: For every PPT adversary  $A = (A_1, A_2)$ , there exists a PPT simulator  $S$  such that the following holds.

$$\left| \Pr \left[ \begin{array}{l} A_2(ck, c, \rho) = 1 \wedge \\ \text{PC.Check}(ck, c, d, x, f(x), \rho) = 1 \end{array} \middle| \begin{array}{l} ck \leftarrow \text{PC.Setup}(1^\lambda, D) \\ (d, f, x) \leftarrow A_1(ck) \\ (c, \rho) \leftarrow S(ck, x, f(x)) \end{array} \right] - \Pr \left[ \begin{array}{l} A_2(ck, c, \rho) = 1 \wedge \\ \text{PC.Check}(ck, c, d, x, f(x), \rho) = 1 \end{array} \middle| \begin{array}{l} ck \leftarrow \text{PC.Setup}(1^\lambda, D) \\ (d, f, x) \leftarrow A_1(ck) \\ (c, \delta) \leftarrow \text{PC.Com}(ck, d, f) \\ (y, \rho) \leftarrow \text{PC.Eval}(x, \delta) \end{array} \right] \right| \leq \text{negl}(\lambda)$$

## 2.4. Interactive Argument of Knowledge

We define an interactive argument of knowledge with the honest verifier zero-knowledge (HVZK) property as follows.

**Definition 2** (Interactive Argument of Knowledge). Let  $\Pi = (\text{Setup}, P, V)$  be an interactive protocol between a prover  $P$  and a verifier  $V$ .  $\Pi$  is called an argument of knowledge for a relation  $R$  if it satisfies the following properties.

- **Completeness**: For all PPT adversary  $A$ , the following holds.

$$\Pr \left[ \begin{array}{l} \langle P(pp, x, w), V(pp, x) \rangle = 1 \vee \\ (x, w) \notin R \end{array} \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (x, w) \leftarrow A(pp) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

- **Knowledge Soundness**: For every PPT adversary  $A = (A_1, A_2)$ , there exists a PPT extractor  $E$  such that, given oracle access to  $A$ , the following holds.

$$\Pr \left[ \begin{array}{l} \langle A_2(pp, st, x), V(pp, x) \rangle = 1 \wedge \\ (x, w) \notin R \end{array} \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (st, x) \leftarrow A_1(pp) \\ w \leftarrow E^A(pp, x) \end{array} \right] \leq \text{negl}(\lambda)$$

$\Pi$  is called honest verifier zero-knowledge (HVZK) if the following holds.

- **HVZK**: For every PPT adversary  $A = (A_1, A_2)$ , there exists a PPT simulator  $S$  such that the following holds, where  $\text{View}$  outputs the verifier's view.

$$\left| \Pr \left[ \begin{array}{l} A_2(ck, \text{view}) = 1 \wedge \\ (x, w) \in R \end{array} \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (x, w) \leftarrow A_1(pp) \\ \text{view} \leftarrow S(pp, x) \end{array} \right] - \Pr \left[ \begin{array}{l} A_2(ck, \text{view}) = 1 \wedge \\ (x, w) \in R \end{array} \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (x, w) \leftarrow A_1(pp) \\ \text{view} \leftarrow \text{View}(P(pp, x, w), V(pp, x)) \end{array} \right] \right| \leq \text{negl}(\lambda)$$

Additionally,  $\Pi$  is called public coin if all messages from the honest verifier can be computed as a deterministic function of a random public input.

If an interactive argument of knowledge is public-coin, it can be transformed into a non-interactive version using the Fiat-Shamir transform [44]. Additionally, if it satisfies the HVZK property, the resulting non-interactive argument becomes a non-interactive zero-knowledge argument. Thus, the HVZK property is sufficient for this work, even though there is a more general version of the ZK property that includes the case of a malicious verifier.

Next, we review the definition of a polynomial interactive oracle proof (PIOP) in [24, 25], which is a special class of interactive arguments of knowledge. In the following definition, we restrict polynomials to be univariate, but it can be generalized to the multivariate case, as defined in [45].

**Definition 3** (Polynomial Interactive Oracle Proof). Let  $\Pi = (\text{Setup}, P, V)$  be an interactive public coin argument of knowledge for a relation  $R$ .  $\Pi$  is called a polynomial interactive oracle proof (PIOP), which satisfies the followings.

- Every message from the prover is a polynomial oracle  $([f], d)$ , where  $f \in \mathbb{Z}_p[X]$  of degree  $\leq d$ .
- Every message from the verifier is a random challenge.
- At the end of the protocol, the verifier receives oracle access to polynomial evaluations at any points.

$\Pi$  is called a honest verifier zero-knowledge PIOP if it is an HVZK argument of knowledge, where  $\text{View}$  outputs the messages from the verifier and the responses to polynomial evaluation queries. The complexity of PIOP is measured as follows:

- **Prover complexity**: The sum of the runtime of the PIOP prover
- **Verifier complexity**: The sum of the runtime of the PIOP verifier

- **Query complexity:** The number of queries the verifier performs to the oracles.
- **Size of proof oracles:** The length of the transmitted polynomials.
- **Size of the witness:** The length of the witness polynomial.

In [24], Chiesa et al. formalize how a PIOP can be compiled into an argument of knowledge using a PCS. In short, the compilation is done by replacing all the oracle polynomials in the PIOP with commitments from the PCS, and then attaching evaluation proofs from the PCS for each polynomial query in the PIOP. The complexity of the resulting argument of knowledge can be described as follows.

**Theorem 1** (Theorem 8.1 [24]). *Let  $\Pi$  be a PIOP for a relation  $R$  and PC be a polynomial commitment scheme. Then, there exists a public coin argument of knowledge  $\Pi'$  for  $R$  with the following complexity.*

- **Prover complexity:** The sum of the runtime of the PIOP prover, the time to commit polynomials in PC, and the time to produce evaluation proofs for oracle queries in PC.
- **Verifier complexity:** The sum of the runtime of the PIOP verifier, the time to verify evaluation proofs in PC.
- **Proof size:** The sum of the messages from the PIOP verifier, commitments size in PC, and evaluation proof size in PC.

Additionally, if  $\Pi$  is HVZK and PC is hiding, then  $\Pi'$  is HVZK.

## 2.5. Multi-group Homomorphic Encryption

Multi-group homomorphic encryption (MGHE) is a variant of homomorphic encryption that allows homomorphic computation over encrypted data from multiple entities. In MGHE, a fixed set of entities is called a group, where each member knows each other prior to computation but does not trust each other. MGHE can be interpreted as a generalization of HE, MPHE, and MKHE, because HE corresponds to a single group with one entity, MPHE to a single group with multiple entities, and MKHE to multiple groups, each consisting of a single entity. The basic pipeline of MGHE consists of the setup, encryption, evaluation, and decryption phases, as illustrated in Fig. 1.

In the setup phase, each member of a group generates public keys in a distributed manner and then constructs joint public keys with group members through a key aggregation protocol. In the encryption phase, each entity encrypts its input using its group's public key and sends it to an evaluator. In the evaluation phase, the evaluator performs homomorphic operations not only with ciphertexts from the same group but also with ciphertexts from different groups. In this case, access control of intermediate ciphertexts are updated in an on-the-fly manner whenever encrypted data from a new group is used. In the decryption phase, each member of the associated groups of the output ciphertext generates partial decryption in a distributed manner, and decryption is completed by aggregating these partial decryptions.

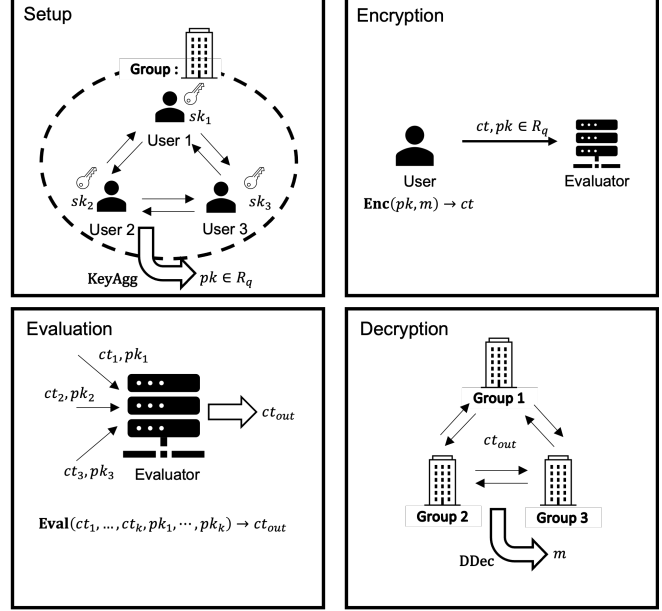


Figure 1. An illustration of MGHE pipeline.

In this work, we focus on the multi-group variant of the BGV scheme [46, 26], following the constructions in [11]. The structure of ciphertexts changes over time as the number of associated groups increases, especially when homomorphic operations occur between different groups. Initially, a fresh ciphertext consists of two polynomials, but its length grows to  $k + 1$  polynomials when the number of associated groups is  $k$ . Since the complexity of homomorphic operations grows with  $k$ , it is beneficial to keep the number of groups small by using a key aggregation protocol. For more details, such as algorithms for homomorphic operations, we refer to the relevant literature [11, 10]. Below, we describe the algorithms for setup, key generation, encryption, and decryption based on the residue number system (RNS) implementation [47]. Its security is based on the RLWE problem and the CRS model.

- **MGBFV.Setup**( $1^\lambda, N$ )  $\rightarrow$  pp: Given a security parameter  $\lambda$  and a ring degree  $N$ , choose a ciphertext modulus  $q = \prod_{i=0}^{\ell-1} q_i$ , where the  $q_i$ 's are distinct prime numbers, a gadget dimension  $\delta \mid \ell$ , a key distribution  $\chi$  over  $R$ , a noise distribution  $\psi$  over  $R$ , a set of automorphisms  $\Phi \subseteq \text{Aut}(R)$ , an upper bound  $B_{\text{DD}}$  for distributed decryption, and generate common random strings  $\mathbf{u}_{\text{ek}} \leftarrow \mathcal{U}(R_q)$ ,  $\vec{\mathbf{u}}_{\text{rlk}} \leftarrow \mathcal{U}(R_q^{2\delta})$ , and  $\vec{\mathbf{u}}_\varphi \leftarrow \mathcal{U}(R_q^\delta)$ . Return a public parameter  $\text{pp} = (R, q, t, \chi, \psi, \Phi, B_{\text{DD}}, \mathbf{u}_{\text{ek}}, \vec{\mathbf{u}}_{\text{rlk}}, \{\vec{\mathbf{u}}_\varphi\}_{\varphi \in \Phi})$ .
- **MGBFV.KeyGen**(pp)  $\rightarrow$  (sk, pk): Given a public parameter pp, generate a secret key sk and a public key pk as follows, where  $\text{pk} = (\text{ek}, \text{rlk}, \{\text{atk}_\varphi\}_{\varphi \in \Phi})$ . A vector  $\vec{g} = (q/\prod_{i=0}^{\ell/\delta-1} q_i, \dots, q/\prod_{i=\ell-\ell/\delta}^{\ell-1} q_i) \in R_q^\delta$  is called the gadget vector.
  - **Secret key:** Sample  $s \leftarrow \chi$ , and return  $\text{sk} = s$ .
  - **Encryption key:** Sample  $\mathbf{e}_{\text{ek}} \leftarrow \psi$ , compute  $\mathbf{p} =$

$-\mathbf{u}_{\text{ek}}\mathbf{s} + \mathbf{e}_{\text{ek}} \pmod{q}$ , and return  $\text{ek} = \mathbf{p} \in R_q$ .

- **Relinearization key:** Sample  $\mathbf{f}_{\text{rlk}} \leftarrow \chi$ ,  $\vec{\mathbf{e}}_{\text{rlk}} \leftarrow \psi^{3\delta}$ , and compute the followings, where  $\vec{\mathbf{u}}_{\text{rlk}} = \vec{\mathbf{u}}_{\text{rlk},0} \parallel \vec{\mathbf{u}}_{\text{rlk},1}$  and  $\vec{\mathbf{e}}_{\text{rlk}} = \vec{\mathbf{e}}_{\text{rlk},0} \parallel \vec{\mathbf{e}}_{\text{rlk},1} \parallel \vec{\mathbf{e}}_{\text{rlk},2}$

$$\begin{aligned}\vec{\mathbf{r}}_0 &= -\mathbf{s} \cdot \vec{\mathbf{u}}_{\text{rlk},0} + \vec{\mathbf{e}}_{\text{rlk},0} \pmod{q} \\ \vec{\mathbf{r}}_1 &= -\mathbf{f}_{\text{rlk}} \cdot \vec{\mathbf{u}}_{\text{rlk},0} + \mathbf{s} \cdot \vec{\mathbf{g}} + \vec{\mathbf{e}}_{\text{rlk},1} \pmod{q} \\ \vec{\mathbf{r}}_2 &= -\mathbf{s} \cdot \vec{\mathbf{u}}_{\text{rlk},1} - \mathbf{f}_{\text{rlk}} \cdot \vec{\mathbf{g}} + \vec{\mathbf{e}}_{\text{rlk},2} \pmod{q}\end{aligned}$$

Return  $\text{rlk} = (\vec{\mathbf{r}}_0, \vec{\mathbf{r}}_1, \vec{\mathbf{r}}_2) \in R_q^{3\delta}$ .

- **Automorphism keys:** For  $\varphi \in \Phi$ , sample  $\vec{\mathbf{e}}_\varphi \leftarrow \psi^\delta$ , compute  $\vec{\mathbf{a}}_\varphi = -\mathbf{s} \cdot \vec{\mathbf{u}}_\varphi + \varphi(\mathbf{s}) \cdot \vec{\mathbf{g}} + \vec{\mathbf{e}}_\varphi \pmod{q}$ , and return  $\text{atk}_\varphi = \vec{\mathbf{a}}_\varphi \in R_q^\delta$ .
- **MGBFV.AggKey**( $\{\text{pk}^{(i)}\}_{i \in I}$ )  $\rightarrow \text{pk}^{(I)}$ : Given a collection of public keys of a group  $I$ , return the aggregated public key  $\text{pk}^{(I)} = (\text{ek}^{(I)}, \text{rlk}^{(I)}, \{\text{atk}_\varphi^{(I)}\}_{\varphi \in \Phi})$ , where  $\text{ek}^{(I)} = \sum_{i \in I} \text{ek}^{(i)} \pmod{q}$ ,  $\text{rlk}^{(I)} = \sum_{i \in I} \text{rlk}^{(i)} \pmod{q}$ , and  $\text{atk}_\varphi^{(I)} = \sum_{i \in I} \text{atk}_\varphi^{(i)} \pmod{q}$ .
- **MGBFV.Enc**( $\text{ek}^{(I)}, \mathbf{m}$ )  $\rightarrow \text{ct} = (\mathbf{c}_0, \mathbf{c}_I)$ : Given an encryption key  $\text{ek}^{(I)} = \mathbf{p}$  of a group  $I$ , a plaintext  $\mathbf{m} \in R_t$ , sample  $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \psi$ ,  $\mathbf{f} \leftarrow \chi$ , and return  $\text{ct} = \mathbf{f} \cdot (\mathbf{p}, \mathbf{u}_{\text{ek}}) + (\lfloor q/t \rfloor \cdot \mathbf{m} + \mathbf{e}_0, \mathbf{e}_1) \pmod{q}$ .
- **MGBFV.DDec**( $\text{sk}^{(i)}, \text{ct}$ )  $\rightarrow \mathbf{d}^{(i)}$ : Given a ciphertext  $\text{ct} = (\mathbf{c}_0, \{\mathbf{c}_{I_j}\}_{0 \leq j < k})$  and the entity  $i$ 's secret key  $\text{sk}^{(i)} = \mathbf{s}$ , sample  $\mathbf{e} \leftarrow \mathcal{U}(R_{B_{\text{DB}}})$ , and return a partial decryption  $\mathbf{d}^{(i)} = \mathbf{c}_{I_j} \mathbf{s} + \mathbf{e} \pmod{q}$ , where  $i \in I_j$ .
- **MGBFV.AggDec**( $\text{ct}, \{\mathbf{d}^{(i)}\}_{i \in \cup_{j=0}^{k-1} I_j}$ )  $\rightarrow \mathbf{m}$ : Given a ciphertext  $\text{ct} = (\mathbf{c}_0, \{\mathbf{c}_{I_j}\}_{0 \leq j < k})$  and a collection of partial decryptions from all engaged entities, return a plaintext  $\mathbf{m} = \left\lfloor \frac{t}{q} (\mathbf{c}_0 + \sum_{i \in \cup_{j=0}^{k-1} I_j} \mathbf{d}^{(i)}) \right\rfloor \pmod{t}$ .

### 3. Review of Univariate PIOP

In this section, we review the PIOPs from [28], which aim to prove the satisfiability of rank-1 constraint systems (RICS) by introducing two subprotocols: the row check PIOP and linear check PIOP. The linear check PIOP was later improved in [24], which reduced verification complexity by introducing a preprocessing method. However, we choose not to use this improved version, as it is only beneficial for verifying multiple proofs for the same constraint, which is not our case, and it also increases the prover's computational overhead. In the rest of this section, we first review the polynomial encoding method and then present the details of the row check PIOP and linear check PIOP. We note that all proofs are deferred to Appendix A.

#### 3.1. Polynomial Encoding

Both the row check and linear check PIOP are based on polynomial encoding, which maps input witness vectors into univariate polynomials. Let  $\mathbb{F}$  be a finite field, and let  $H = \{h_0, \dots, h_{n-1}\} \subseteq \mathbb{F}$  such that  $|H| = n$ . We define a polynomial encoding as follows.

- **Ecd**( $\vec{w}$ )  $\rightarrow \mathbf{w}$ : Given a vector  $\vec{w} \in \mathbb{F}^n$ , output the polynomial  $\mathbf{w} \in \mathbb{F}^{<n}[X]$  such that  $\mathbf{w}(h_i) = w_i$  for  $0 \leq i < n$ .

We note that such  $\mathbf{w}$  is uniquely determined due the degree upper bound. To achieve zero-knowledgeness, encoding procedure is often randomized. Let  $L > n$  be an integer, then the randomized encoding is defined as follows.

- **REcd**( $\vec{w}$ )  $\rightarrow \hat{\mathbf{w}}$ : Given a vector  $\vec{w} \in \mathbb{F}^n$ , uniformly sample a polynomial  $\hat{\mathbf{w}} \in \mathbb{F}^{<L}[X]$  such that  $\hat{\mathbf{w}}(h_i) = w_i$  for  $0 \leq i < n$ .

We note that there are multiple candidates for  $\hat{\mathbf{w}}$ . If we sample such  $\hat{\mathbf{w}}$  uniformly, then the evaluation results  $\hat{\mathbf{w}}(\alpha)$  for  $\alpha \notin H$  are independent of  $\vec{w}$  up to  $L-n$  evaluations due to the bounded independence. For details, we refer to [28]. In the following PIOPs, this property is crucial for achieving the HVZK property. We denote the set of candidates for the evaluation points as  $C \subseteq \mathbb{F}$ , which satisfies  $C \cap H = \emptyset$ .

#### 3.2. PIOP for RICS

**Row Check.** The row check PIOP is for proving the arithmetic relation over  $\mathbb{F}$  for each row of input witness vectors. The arithmetic relation can be described as a multivariate polynomial, where the number of variables is equal to the number of input witness vectors. Let  $\mathbf{z}_H \in \mathbb{F}^{<n}[X]$  be the vanishing polynomial of  $H$ , which satisfies  $\mathbf{z}_H(h_i) = 0$  for  $0 \leq i < n$ . Then, the row check PIOP  $\Pi_{\text{RC}}$  is defined as in Fig. 2. Its core idea is to utilize the property of the vanishing polynomial, and zero-knowledgeness is assured by the randomized encoding method. The detailed analysis of  $\Pi_{\text{RC}}$  is as follows.

**Theorem 2** (Definition 4.9 [28]). *Let  $\mathcal{C}$  be a  $k$ -ary polynomial of degree  $d$ ,  $U$  be the number of non-zero terms of  $\mathcal{C}$ , and  $\vec{a}_0, \dots, \vec{a}_{k-1} \in \mathbb{F}^n$  be witness. Then,  $\Pi_{\text{RC}}(\mathcal{C}; \vec{a}_0, \dots, \vec{a}_{k-1})$  is an HVZK PIOP with the following complexity.*

- The soundness error is  $O\left(\frac{(k+d)L}{|C|}\right)$
- The prover time is  $O(UL(1 + d \log(Ld) \log d))$
- The verifier time is  $O(Ud + \log n)$
- The query complexity is  $k+1$ . The total number of distinct query points is 1.
- The size of proof oracles is  $kL + d(L-1) - n + 1$ ,  $k$  polynomials of degree  $L-1$  and 1 polynomial of degree  $d(L-1) - N$
- The size of witness is  $kn$ ,  $k$  vectors of length  $n$ .

**Linear Check.** The linear check PIOP proves the linear relationship between input witness vectors, which can be described as a matrix-vector multiplication with a fixed matrix  $M$ . The linear check PIOP  $\Pi_{\text{LC}}$  is defined as in Fig. 3. It is based on the amortized univariate sumcheck protocol in [28], which requires  $H$  to be a multiplicative subgroup of  $\mathbb{F}$ . To achieve zero-knowledgeness, a mask polynomial  $\mathbf{g}$  is needed, as the randomized encoding alone is insufficient. The prover and verifier complexity is primarily affected by the matrix-vector multiplication  $\vec{w} = M^T \vec{v}$ . Below, we provide its detailed analysis.

**Theorem 3** (Theorem 6.2, Lemma 5.10 [28]).  *$\Pi_{\text{LC}}$  is an HVZK PIOP with the following complexity, where  $K$  is the number of non-zero entries in the input matrix  $M$ .*

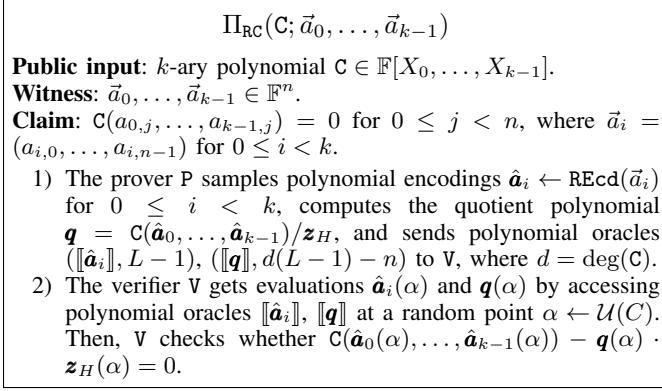


Figure 2. PIOP for row check

- The soundness error is  $O\left(\frac{k(L+n)}{|\mathbb{C}|}\right)$
- The prover time is  $O(K + L \log L + kL)$
- The verifier time is  $O(K + k)$
- The query complexity is  $2k + 3$  and the total number of distinct query points is 1.
- The size of proof oracles is  $2kL + 2L + 2n - 3$ .  $2k$  polynomials of degree  $L - 1$  and three polynomials of degree  $L+n-2$ ,  $L-2$ , and  $n-2$  respectively. Additionally  $\text{P}$  sends one field elements to verifier.
- The size of witness is  $2kn$ ,  $2k$  vectors of length  $n$ .

## 4. PIOP for MGHE

In this section, we present PIOP for MGHE, which checks validity of public keys, ciphertexts, and partial decryptions. We first demonstrate how we adapt constraints over a polynomial ring to align with the univariate PIOPs in Section 3.2. As described in Section 2.5, ciphertexts and public keys are generated over the polynomial ring  $R_q$ , while the PIOPs in Section 3.2 are designed to handle vectors over a finite field. We bridge this gap using the number-theoretic transform and modulus-switching techniques, enabling us to prove relations over polynomial rings with the univariate PIOP. Based on this approach, we demonstrate how the validity of key generation, encryption, and decryption can be verified within the univariate PIOP framework. We note that all proofs are deferred to Appendix A.

### 4.1. PIOP for Polynomial Ring

**Vector Representation.** To prove the validity of ciphertexts and public keys, we primarily need to verify two types of constraints over the polynomial ring  $R_q$ : the arithmetic relations between polynomials and the boundedness of polynomial coefficients. However, as noted earlier, we need to convert these polynomial constraints into vector constraints to leverage the PIOPs in Section 3.2. To resolve this issue, we use two types of vector representations for polynomials: the coefficient representation and the number-theoretic transform (NTT) representation.

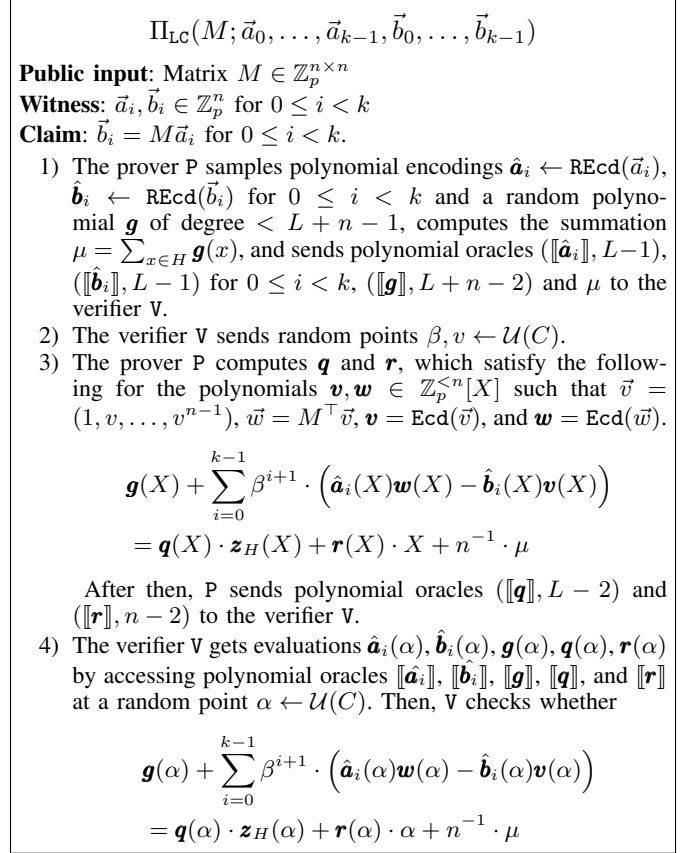


Figure 3. PIOP for batched linear relation check

The coefficient representation is simply a vector of coefficients, which preserves the norm but not the algebraic structure. In contrast, the NTT representation aims to preserve the algebraic structure. If the modulus  $q$  is NTT-friendly, i.e., it has a  $2N$ -th root of unity,  $R_q$  is isomorphic to  $\mathbb{Z}_q^N$  via an isomorphism called NTT, which can be computed in  $O(N \log N)$  complexity. Fortunately, most HE schemes use an NTT-friendly modulus for fast polynomial multiplication, so we can assume  $q$  is NTT-friendly. The output vector of the NTT, which we call the NTT representation, preserves algebraic structure; however, the norm is not preserved in this case. Since we need to prove both arithmetic relations and the boundedness of coefficients, we use both vector representations in our PIOP for MGHE. Below, we summarize each vector representation.

- **Coeff**( $\mathbf{a}$ )  $\rightarrow \vec{a}$ : Given a polynomial  $\mathbf{a} = \sum_{i=0}^{N-1} a_i X^i$ , outputs a vector  $\vec{a} = (a_0, \dots, a_{N-1})$ .
- **NTT**( $\mathbf{a}$ )  $\rightarrow \vec{a}$ : Given a polynomial  $\mathbf{a} = \sum_{i=0}^{N-1} a_i X^i$ , outputs a vector  $\vec{a} = (\mathbf{a}(\xi), \mathbf{a}(\xi^3), \dots, \mathbf{a}(\xi^{2N-1}))$ , where  $\xi$  is a  $2N$ -th root of unity.

**Modulus Switching.** As mentioned in Section 2.5, the modulus  $q$  is a composite number, a product of distinct primes. However, the PIOP in Section 3.2 are designed to support vectors over finite fields, and  $\mathbb{Z}_q$  is not a field. One way to circumvent this issue is to prove polynomial

constraints for each  $R_{q_i}$  by leveraging the isomorphism relation  $R_q \cong \prod_{i=0}^{\ell-1} R_{q_i}$ . However, this significantly degrades performance in the PIOP setting, because each  $q_i$  is too small to achieve negligible soundness error, and it inhibits optimization techniques such as batch evaluation protocols when we compile PIOP into zk-SNARK using PCS.

We resolve this issue using the modulus switching operation, a technique frequently used in RLWE-based HE schemes to adjust the ciphertext modulus. In short, the modulus switching operation maps  $\vec{c} \in R_q^k$  to  $\left\lfloor \frac{q'}{q} \vec{c} \right\rfloor \in R_{q'}^k$  when changing the modulus from  $q$  to  $q'$ . Although it has been used for homomorphic operations, we observe that it can also be utilized to prove the validity of ciphertexts and public keys over a PIOP-friendly modulus, rather than over the composite modulus  $q$ . To this end, we modified the pipeline for MGHE, as shown in Fig. 4, to take advantage of modulus switching in constructing PIOP for MGHE.

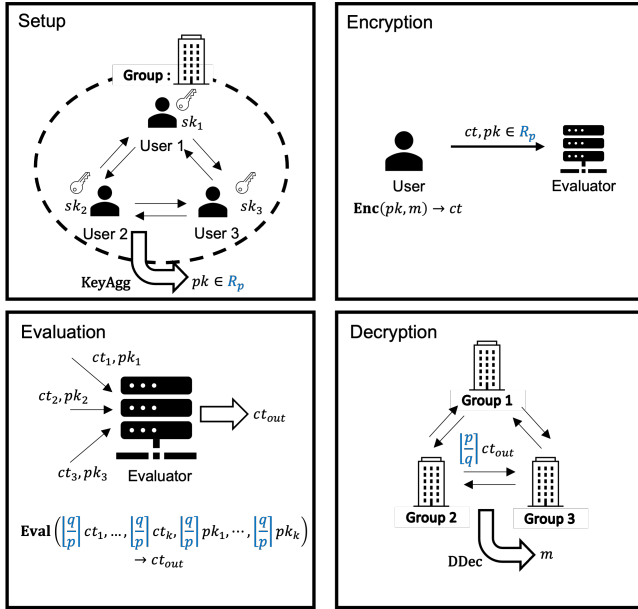


Figure 4. An illustration of modified MGHE pipeline.

In the setup and encryption phase, public keys and ciphertexts are generated over  $R_p$ , where  $p$  is an NTT-friendly prime number such that  $p \approx q$ . During the evaluation phase, an evaluator switches the modulus of the public keys and ciphertexts from  $p$  to  $q$  for efficient homomorphic computations. At the end of the evaluation phase, the evaluator switches the modulus of the output ciphertext from  $q$  to  $p$ . Finally, the decryption phase proceeds with the output ciphertext over  $R_p$ . This modification allows us to construct PIOP for MGHE over the polynomial ring  $R_p$ .

**Generalized Row Check.** We now demonstrate how to prove arithmetic constraints over a polynomial ring using a PIOP. Thanks to modulus switching, we can assume that

witness polynomials and their arithmetic relations are represented in  $R_p$ . Then, by applying the NTT representation, the arithmetic constraints over the polynomial ring  $R_p$  are converted to arithmetic constraints over the product ring  $\mathbb{Z}_p^N$ . Thus, for our purpose, it suffices to prove the arithmetic constraints over the product ring  $\mathbb{Z}_p^N$  using a PIOP. However, existing PIOP, such as the row check, do not cover this case. We recall that the row check PIOP can only prove a single, identical arithmetic constraint over  $\mathbb{Z}_p$  for each row of witness vectors. In contrast, in our case, we need to prove different constraints for each row of witness vectors, as the constraints are defined over  $\mathbb{Z}_p^N$ .

To address this issue, we devise a new PIOP called the generalized row check, which is a variant of the row check PIOP. The detailed process is presented in Fig. 5. In the protocol, we represent an arithmetic constraint as a multivariate polynomial  $\vec{C}$  whose coefficients are in  $\mathbb{Z}_p^N$ , and multiplications are performed using Hadamard products. During the protocol, the goal is to prove the satisfiability of  $\vec{C}(\vec{a}_0, \dots, \vec{a}_{k-1}) = \vec{0}$  for witness vectors  $\vec{a}_0, \dots, \vec{a}_{k-1}$ . These witness vectors are encoded to polynomials via REcd, where we set  $H = \{1, \xi^2, \dots, \xi^{2N-2}\}$  for a  $2N$ -th root of unity  $\xi$  in  $\mathbb{Z}_p$ , so that  $H$  forms a multiplicative subgroup of  $\mathbb{Z}_p$ . Next, we observe that each coefficient of  $\vec{C}$  can be encoded as polynomials via the map Ecd so that  $\vec{C}$  is converted into a multivariate polynomial  $\mathbf{C}$  whose coefficients are in  $\mathbb{Z}_p[X]$ . Then, we obtain the following equivalences by the property of polynomial encoding and the vanishing polynomial  $z_H$ .

$$\begin{aligned} \vec{C}(\vec{a}_0, \dots, \vec{a}_{k-1}) = \vec{0} &\iff \mathbf{C}(\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_{k-1})(h) = 0, \forall h \in H \\ &\iff z_H \mid \mathbf{C}(\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_{k-1}) \end{aligned}$$

Hence, it suffices to check the last divisibility, as described in  $\Pi_{\text{GRC}}$ . The detailed analysis of  $\Pi_{\text{GRC}}$  is as follows.

**Theorem 4 (Generalized row check).** *Let  $\vec{C}$  be a  $k$ -ary polynomial with degree  $d$ ,  $U$  non-zero terms and  $\vec{a}_0, \dots, \vec{a}_{k-1}$  be  $N$ -length vectors over  $\mathbb{Z}_p$ . Then,  $\Pi_{\text{GRC}}$  is an HVZK PIOP with the following complexity where  $L = O(N)$ .*

- The soundness error is  $O\left(\frac{(k+d)N}{|C|}\right)$ .
- The prover time is  $O(UN(d+1) \log(Nd+N) \log(d+1))$ .
- The verifier time is  $O(UN + Ud)$ .
- The query complexity is  $k+1$ . The total number of distinct query points is 1.
- The size of proof oracles is  $kL + (d+1)(L-1) - N + 1$ .
- The size of witness is  $kN$ ,  $k$  vectors of length  $N$ .

**Norm Check.** For the validity of ciphertexts and public keys, we also need to prove the boundedness of the coefficients of polynomials, i.e., an upper bound on the norm of the polynomials. Since the coefficient representation converts a polynomial to a vector while preserving the norm, it suffices to construct a PIOP that checks an upper bound on the norm of the input witness vector. To achieve this, we construct a PIOP called the norm check PIOP, which is a combination of several row check PIOPs. The detailed process is illustrated in Fig. 6.



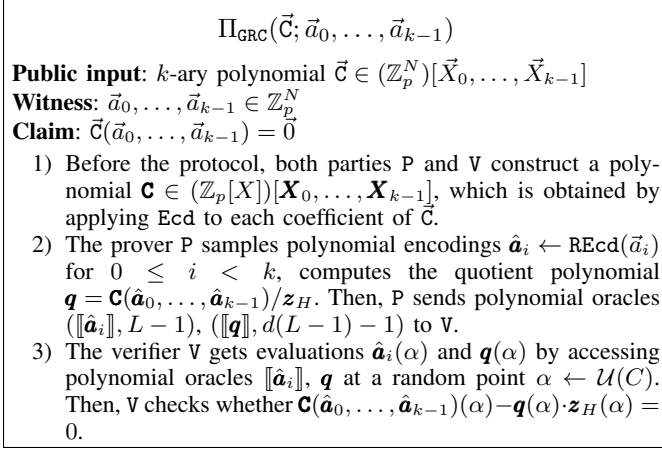


Figure 5. PIOP for generalized row check

The goal of this PIOP is to prove  $\|\vec{a}\|_\infty \leq B$  for a witness vector  $\vec{a} \in \mathbb{Z}_p^N$ . To prove this, we utilize the ternary representation method from [48]. For  $B > 0$ , let  $k = \lfloor \log B \rfloor + 1$  and define  $B_0 = \lceil \frac{B}{2} \rceil$ ,  $B_1 = \lceil \frac{B-B_0}{2} \rceil$ ,  $B_2 = \lceil \frac{B-B_0-B_1}{2} \rceil, \dots, B_{k-1} = 1$ , and  $B_k = 0$ . If  $\|\vec{a}\|_\infty \leq B$  holds, it can be decomposed as  $\vec{a} = \sum_{i=0}^{k-1} B_i \cdot \vec{a}_i$ , where each  $\vec{a}_i$  is a ternary vector, i.e., each entry is either -1, 0, or 1. Then, the statement about an upper bound on the norm can be translated into the following arithmetic constraints:  $\vec{a} - \sum_{i=0}^{k-1} B_i \cdot \vec{a}_i = \vec{0}$  and  $\vec{a}_i \circ (\vec{a}_i - \vec{1}) \circ (\vec{a}_i + \vec{1}) = \vec{0}$  for  $0 \leq i < k$ , which can be checked through row check PIOPs, as illustrated in Fig. 6. Below, we provide an analysis of the norm check PIOP  $\Pi_{\text{NC}}$ .

**Theorem 5** (Norm check).  $\Pi_{\text{NC}}$  is an HVZK PIOP with the following complexity where  $k = \lfloor \log B \rfloor + 1$  and  $L = O(N)$ .

- The soundness error is  $O\left(\frac{kN}{|\vec{C}|}\right)$ .
- The prover time is  $O(kN \log N)$ .
- The verifier time is  $O(k + \log N)$ .
- The query complexity is  $2k + 2$ . The total number of distinct query points is 1.
- The size of proof oracles is  $4kL - kN + 2L - N - 2k \lfloor \mathbb{Z}_p \rfloor$ .
- The size of witness is  $N$ , a vector of length  $N$ .

## 4.2. PIOP for Key Generation

Based on the generalized row check and the norm check PIOPs, we can prove the arithmetic constraints and boundedness of coefficients for polynomials in  $R_p$ . As noted previously, our basic strategy is to generate public keys and ciphertexts in  $R_p$  and prove their validity using univariate PIOP over  $\mathbb{Z}_p^N$ . This naturally raises the question of how to generate valid public keys in  $R_p$ , given that the key generation algorithm in Section 2.5 is described based on  $R_q$ . To address this question, we modify MGBFV.Setup and MGBFV.KeyGen as follows to accommodate the modulus change from  $q$  to  $p$ .

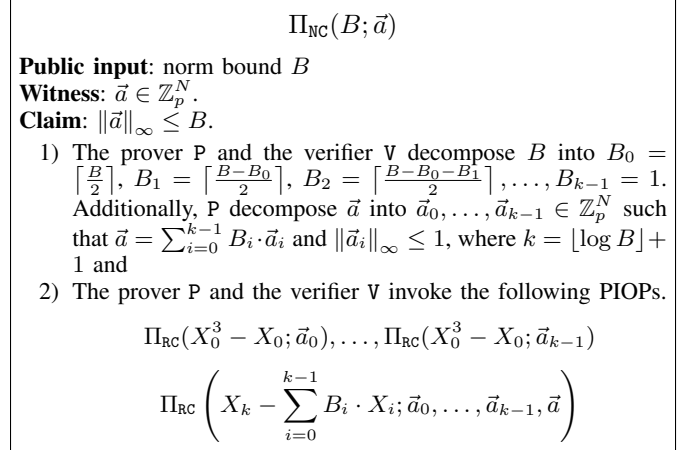


Figure 6. PIOP for norm check

- **Setup:** Choose an NTT-friendly prime number  $p$  such that  $p \approx q$ , and generate common random strings  $\mathbf{u}_{\text{ek}} \leftarrow \mathcal{U}(R_p)$ ,  $\vec{\mathbf{u}}_{\text{rlk}} \leftarrow \mathcal{U}(R_p^{2\delta})$ , and  $\vec{\mathbf{u}}_\varphi \leftarrow \mathcal{U}(R_p^\delta)$ .
- **Encryption key:** Sample  $\mathbf{e}_{\text{ek}} \leftarrow \psi$ , compute  $\mathbf{p} = -\mathbf{u}_{\text{ek}} \mathbf{s} + \mathbf{e}_{\text{ek}} \pmod{p}$ , and return  $\text{ek} = \mathbf{p} \in R_p$ .
- **Relinearization key:** Sample  $\mathbf{f}_{\text{rlk}} \leftarrow \chi$ ,  $\vec{\mathbf{e}}_{\text{rlk}} \leftarrow \psi^{3\delta}$ , and compute the followings, where  $\vec{\mathbf{u}}_{\text{rlk}} = \vec{\mathbf{u}}_{\text{rlk},0} \parallel \vec{\mathbf{u}}_{\text{rlk},1}$  and  $\vec{\mathbf{e}}_{\text{rlk}} = \vec{\mathbf{e}}_{\text{rlk},0} \parallel \vec{\mathbf{e}}_{\text{rlk},1} \parallel \vec{\mathbf{e}}_{\text{rlk},2}$ 

$$\vec{\mathbf{r}}_0 = -\mathbf{s} \cdot \vec{\mathbf{u}}_{\text{rlk},0} + \vec{\mathbf{e}}_{\text{rlk},0} \pmod{p}$$

$$\vec{\mathbf{r}}_1 = -\mathbf{f}_{\text{rlk}} \cdot \vec{\mathbf{u}}_{\text{rlk},0} + \mathbf{s} \cdot \left\lfloor \frac{p}{q} \vec{g} \right\rfloor + \vec{\mathbf{e}}_{\text{rlk},1} \pmod{p}$$

$$\vec{\mathbf{r}}_2 = -\mathbf{s} \cdot \vec{\mathbf{u}}_{\text{rlk},1} - \mathbf{f}_{\text{rlk}} \cdot \left\lfloor \frac{p}{q} \vec{g} \right\rfloor + \vec{\mathbf{e}}_{\text{rlk},2} \pmod{p}$$

Return  $\text{rlk} = (\vec{\mathbf{r}}_0, \vec{\mathbf{r}}_1, \vec{\mathbf{r}}_2) \in R_p^{3\delta}$ .
- **Automorphism keys:** For  $\varphi \in \Phi$ , sample  $\vec{\mathbf{e}}_\varphi \leftarrow \psi^\delta$ , compute  $\vec{\mathbf{a}}_\varphi = -\mathbf{s} \cdot \vec{\mathbf{u}}_\varphi + \varphi(\mathbf{s}) \cdot \left\lfloor \frac{p}{q} \vec{g} \right\rfloor + \vec{\mathbf{e}}_\varphi \pmod{p}$ , and return  $\text{atk}_\varphi = \vec{\mathbf{a}}_\varphi \in R_p^\delta$ .

Based on these modifications, we present a PIOP for MGBFV.KeyGen in Fig. 7, where we assume samples from  $\chi$  and  $\psi$  are bounded by  $B$ . Below, we demonstrate how to prove the validity of each type of keys.

**Secret Key.** For a secret key  $\mathbf{s}$ , we can sample it directly from a key distribution  $\chi$  without modification, as only its smallness is essential, and the ciphertext modulus does not impact this property. The smallness of the secret key is crucial for ensuring the correctness of homomorphic operations, so we need to prove its boundedness to prevent malicious behavior. This can be proven through the norm check PIOP by putting coefficient representation  $\vec{s}$  of  $\mathbf{s}$ . However, in the remainder of each protocol, we use the NTT representation  $\underline{\vec{s}}$  of  $\mathbf{s}$  since we typically prove the arithmetic constraints. Thus, we also need to prove the relation between  $\underline{\vec{s}}$  and  $\vec{s}$ , which can be described as a linear relation  $\underline{\vec{s}} = T\vec{s}$ , where

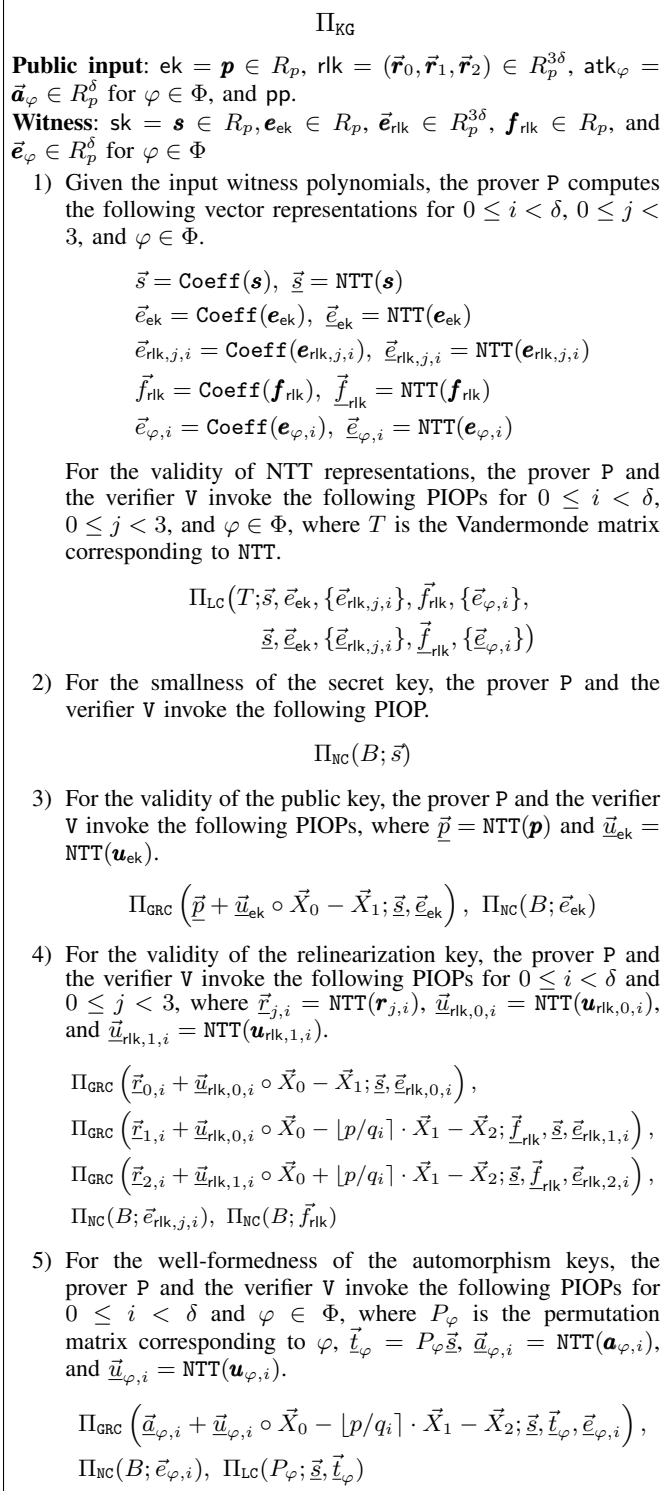


Figure 7. PIOP for MGBFV.KeyGen

$T$  is the Vandermonde matrix with the following entries.

$$T = \begin{bmatrix} 1 & \xi & \cdots & \xi^{N-1} \\ 1 & \xi^3 & \cdots & \xi^{3(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \xi^{2N-1} & \cdots & \xi^{(2N-1)(N-1)} \end{bmatrix}$$

This can be proved through the linear check PIOP. However, in the analysis in Theorem 3 of  $\Pi_{\text{LC}}$ , the prover and verifier complexity is affected by the number of non-zero entries of  $T$ , due to the computation of  $T^\top \vec{v}$  for a random vector  $\vec{v}$ . Since  $T$  is a dense matrix, a naive matrix-vector multiplication algorithm results in  $O(N^2)$  complexity. To resolve this issue, we leverage the following relation, where  $J$  is the backward identity and  $\text{iNTT}$  is the inverse number-theoretic transform.

$$T^\top \vec{v} = N \cdot \text{iNTT}(J\vec{v})$$

The computation  $J\vec{v}$  requires  $O(N)$  complexity due to the sparsity of  $J$ , and  $\text{iNTT}$  can be computed in  $O(N \log N)$  complexity, similar to the NTT operation. As a result, the linear check PIOP for  $\vec{\mathbf{s}} = T\vec{\mathbf{s}}$  requires  $O(N \log N)$  complexity for both the prover and the verifier, rather than  $O(N^2)$ .

**Encryption Key.** For an encryption key  $\text{ek} = \mathbf{p}$ , it can be easily adapted to  $R_p$  by sampling the CRS  $\mathbf{u}_{\text{ek}}$  from  $\mathcal{U}(R_p)$ , and computing  $\mathbf{p} = -\mathbf{u}_{\text{ek}}\mathbf{s} + \mathbf{e}_{\text{ek}}$  over  $R_p$ . For the correctness of encryption  $\text{MGBFV.Enc}$ , the smallness of both  $\mathbf{s}$  and  $\mathbf{e}_{\text{ek}}$  is crucial. Therefore, we need to prove not only the arithmetic relation between  $\mathbf{p}$  and  $\mathbf{s}, \mathbf{e}_{\text{ek}}$ , but also the smallness of  $\mathbf{e}_{\text{ek}}$ . The arithmetic relation can be proven using the generalized row check by taking the witness  $\mathbf{s}, \mathbf{e}_{\text{ek}}$  as NTT representations. For the smallness of the noise term  $\mathbf{e}_{\text{ek}}$ , we use a similar approach as with the secret key: applying the norm check PIOP for the coefficient representation and proving the linear relation between the NTT and coefficient representations.

**Relinearization Key.** For a relinearization key  $\text{rlk} = (\vec{\mathbf{r}}_0, \vec{\mathbf{r}}_1, \vec{\mathbf{r}}_2)$ , we need several modifications to cope with  $R_p$ . First, we sample the CRS  $\mathbf{u}_{\text{rlk}}$  from  $\mathcal{U}(R_p^{2\delta})$ , similar to the encryption key case. Next, we replace the gadget vector  $\vec{g} \in R_q^\delta$  with its rescaled version  $\lfloor \frac{p}{q} \vec{g} \rfloor \in R_p^\delta$ , which can be interpreted as applying a modulus switching to adjust the gadget vector for the modulus  $p$ . Finally, we need to check whether  $\text{rlk}$  remains valid after modulus switching to  $q$ . If we apply modulus switching on  $\text{rlk}$ , we obtain  $\lfloor \frac{q}{p} \text{rlk} \rfloor \in R_q^{3\delta}$ . The validity of  $\lfloor \frac{q}{p} \text{rlk} \rfloor$  can be proven as follows.

**Lemma 1.** For  $\lfloor \frac{q}{p} \text{rlk} \rfloor \in R_q^{3\delta}$ , the following holds, where  $\|\vec{\mathbf{e}}'_{\text{rlk}}\|_\infty \leq \frac{q}{p} \|\vec{\mathbf{e}}_{\text{rlk}}\|_\infty + \frac{1}{2}(q + 2BN)$ .

$$\begin{aligned} \left\lfloor \frac{q}{p} \vec{\mathbf{r}}_0 \right\rfloor &= -\mathbf{s} \cdot \left\lfloor \frac{q}{p} \vec{\mathbf{u}}_{\text{rlk},0} \right\rfloor + \vec{\mathbf{e}}'_{\text{rlk},0} \pmod{q} \\ \left\lfloor \frac{q}{p} \vec{\mathbf{r}}_1 \right\rfloor &= -\mathbf{f}_{\text{rlk}} \cdot \left\lfloor \frac{q}{p} \vec{\mathbf{u}}_{\text{rlk},0} \right\rfloor + \mathbf{s} \cdot \vec{g} + \vec{\mathbf{e}}'_{\text{rlk},1} \pmod{q} \\ \left\lfloor \frac{q}{p} \vec{\mathbf{r}}_2 \right\rfloor &= -\mathbf{s} \cdot \left\lfloor \frac{q}{p} \vec{\mathbf{u}}_{\text{rlk},1} \right\rfloor - \mathbf{f}_{\text{rlk}} \cdot \vec{g} + \vec{\mathbf{e}}'_{\text{rlk},2} \pmod{q} \end{aligned}$$

Thus,  $\lfloor \frac{q}{p} \text{rlk} \rfloor$  is a valid relinearization key in modulus  $q$  with respect to the CRS  $\lfloor \frac{q}{p} \vec{\mathbf{u}}_{\text{rlk}} \rfloor \in R_q^{2\delta}$ . Thus, for the validity of the relinearization key, it suffices to check the arithmetic relation for  $\text{rlk}$  and the smallness of  $\mathbf{s}, \mathbf{f}$ , and  $\vec{\mathbf{e}}_{\text{rlk}}$  as described in our protocol.

**Automorphism Key.** For an automorphism key  $\text{atk} = \vec{a}_\varphi$ , the basic workflow is similar to the relinearization case. The following lemma shows the validity of  $\left\lfloor \frac{q}{p} \text{atk} \right\rfloor \in R_q$  with respect to the CRS  $\left\lfloor \frac{q}{p} \vec{u}_\varphi \right\rfloor \in R_q^\delta$ .

**Lemma 2.** For  $\left\lfloor \frac{q}{p} \text{atk}_\varphi \right\rfloor \in R_q^\delta$ , the following holds, where  $\|\vec{e}'_\varphi\|_\infty \leq \frac{q}{p} \|\vec{e}_\varphi\|_\infty + \frac{1}{2}(\frac{q}{p} + 2BN)$ .

$$\left\lfloor \frac{q}{p} \vec{a}_\varphi \right\rfloor = -\mathbf{s} \cdot \left\lfloor \frac{q}{p} \vec{u}_\varphi \right\rfloor + \varphi(\mathbf{s}) \cdot \vec{g} + \vec{e}'_\varphi \pmod{q}$$

However, an additional step is required to handle the term  $\varphi(\mathbf{s})$ . We first observe that an automorphism  $\varphi$  is of the form  $X \mapsto X^k$  for some odd integer  $k$ . Then, the following holds.

$$\text{NTT}(\varphi(\mathbf{s})) = \text{NTT}(\mathbf{s}(X^k)) = (\mathbf{s}(\xi^k), \mathbf{s}(\xi^{3k}), \dots, \mathbf{s}(\xi^{(2N-1)k}))$$

Let  $\vec{t}_\varphi = \text{NTT}(\varphi(\mathbf{s}))$ , then there exists a permutation matrix  $P_\varphi$  such that  $\vec{t}_\varphi = P_\varphi \vec{s}$  since each  $\mathbf{s}(\xi^{(2i+1)k})$  is the  $\left\lfloor \frac{(2i+1)k \cdot 2N}{2} \right\rfloor$ -th component of  $\vec{s} = \text{NTT}(\mathbf{s})$  for  $0 \leq i < N$ . Therefore, we can show the validity of automorphism keys by proving the linear relation  $\vec{t}_\varphi = P_\varphi \vec{s}$  in addition to arithmetic relation and smallness of witness, as described in the protocol.

**Analysis.** The complexity analysis of  $\Pi_{\text{KG}}$  is as follows.

**Theorem 6** (PIOP for MGBFV.KeyGen).  $\Pi_{\text{KG}}$  is an HVZK PIOP with the following complexity, where  $k = \lfloor \log B \rfloor + 1$ .

- The soundness error is  $O\left(\frac{|\Phi|k\delta L}{|C|}\right)$ .
- The prover time is  $O(|\Phi|\delta N(\log N + k))$ .
- The verifier time is  $O(|\Phi|\delta(N \log N + k))$ .
- The query complexity is  $(5+k)(|\Phi|+3)\delta + 4|\Phi| + 3k + 16$  and the total number of distinct query points is 1.
- The size of proof oracles is  $(kE + 7E + 3|\Phi|)L - 2(E - |\Phi| - 1)N - 3E - 3|\Phi| - 1$   $|\mathbb{Z}_p|$ , where  $E = (|\Phi| + 3)\delta + 3$ . Additionally  $\mathsf{P}$  sends  $1 + |\Phi|$  field elements to verifier.
- The size of witness is  $((|\Phi| + 3)\delta + 3)N$ ,  $(3 + |\Phi|)\delta + 3$  elements in  $R_p$ .

### 4.3. PIOP for Encryption and Decryption

In addition to proving the validity of public keys, we also need to prove the validity of ciphertexts and partial decryptions. We recall that we modified the MGHE pipeline so that encryptions and partial decryptions proceed in  $R_p$ . Therefore, we can verify the validity of ciphertexts and partial decryptions using a generalized row check and a norm check PIOP over  $\mathbb{Z}_p^N$ , similar to the PIOP for public keys.

**Encryption.** To validate ciphertexts, it suffices to check whether they are correctly generated from the encryption algorithm. However, as we change the modulus from  $p$  to  $q$ , the encryption algorithm should be modified to accommodate the change. Thus, we modify MGBFV.Enc as follows.

- **Encryption:** Given an encryption key  $\text{ek}^{(I)} = \mathbf{p} \in R_p$  of a group  $I$ , a plaintext  $\mathbf{m} \in R_t$ , sample  $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \psi$ ,

$\mathbf{f} \leftarrow \chi$ , and return  $\text{ct} = \mathbf{f} \cdot (\mathbf{p}, \mathbf{u}_{\text{ek}}) + (\lfloor p/t \rfloor \cdot \mathbf{m} + \mathbf{e}_0, \mathbf{e}_1) \pmod{p}$ .

In the modified algorithm, we replace the constant  $\lfloor q/t \rfloor$  in MGBFV.Enc with  $\lfloor p/t \rfloor$  and perform all computations in  $R_p$ . Next, we need to prove that  $\text{ct}$  remains valid after modulus switching to  $R_q$ , since evaluation will proceed in  $R_q$ . This can be shown as follows.

**Lemma 3.** For  $\left\lfloor \frac{q}{p} \text{ct} \right\rfloor \in R_q^2$ , the following holds, where  $\|\mathbf{e}'\|_\infty \leq \frac{q}{p} \|\mathbf{e}\|_\infty + \frac{1}{2}(\frac{q}{p} + (t + B|I|)N)$ .

$$\left\lfloor \frac{q}{p} \mathbf{c}_0 \right\rfloor + \left\lfloor \frac{q}{p} \mathbf{c}_I \right\rfloor \cdot \mathbf{s}^{(I)} = \lfloor q/t \rfloor \cdot \mathbf{m} + \mathbf{e}' \pmod{q}$$

Therefore,  $\text{ct}$  remains valid after modulus switching. In Fig. 8, we illustrate a PIOP for verifying the correct execution of MGBFV.Enc, which proves the arithmetic relation for  $\text{ct}$  and the smallness of the randomness terms  $\mathbf{f}$ ,  $\mathbf{e}_0$ , and  $\mathbf{e}_1$  used in encryption. Additionally, we note that the boundedness of  $\mathbf{m}$  should also be checked to ensure that a fresh ciphertext has a small noise. Below, we provide an analysis of  $\Pi_{\text{Enc}}$ .

**Theorem 7** (PIOP for MGBFV.Enc).  $\Pi_{\text{Enc}}$  is an HVZK PIOP with the following complexity, where  $k_1 = \lfloor \log B \rfloor + 1$  and  $k_2 = \lfloor \log t \rfloor + 1$ .

- The soundness error is  $O\left(\frac{(3k_1+k_2)N}{|C|}\right)$
- The prover time is  $O(N(\log N + k_1 + k_2))$
- The verifier time is  $O(N \log N + k_1 + k_2)$
- The query complexity is  $6k_1 + 2k_2 + 17$  and the total number of distinct query points is 1.
- The size of proof oracles is  $(3k_1 + k_2)(4L - N - 2) + 18L - 4N - 5$ . Additionally,  $\mathsf{P}$  sends one field elements to verifier.
- The size of witness is  $4N$ , 4 elements in  $R_p$ .

**Distributed Decryption.** To validate partial decryptions, it suffices to check whether they are correctly generated from the distributed decryption algorithms. First, we need to check the validity of  $\mathbf{s}$  used for partial decryption. This not only verifies the smallness but also checks whether it is the secret key used for generating  $\text{ek}$ .

Next, we verify the bound of the noise added during distributed decryption. This can be done via the norm check PIOP, but directly applying it causes a huge overhead since the size  $B_{\text{DD}}$  can be hundreds of bits. To resolve this issue, we utilize the optimization technique used in [23], where the noise is sampled by an RLWE sample over  $R_{B_{\text{DD}}}$ , rather than directly from  $\mathcal{U}(R_{B_{\text{DD}}})$ . To be precise, we sample the noise  $\mathbf{e}$  by  $\mathbf{u}_{\text{DD}} \mathbf{f}_{\text{DD}} + \mathbf{e}_{\text{DD}} \pmod{B_{\text{DD}}}$ , where  $\mathbf{u}_{\text{DD}}$  is a CRS sampled from  $\mathcal{U}(R_{B_{\text{DD}}})$ ,  $\mathbf{f}_{\text{DD}} \leftarrow \chi$ , and  $\mathbf{e}_{\text{DD}} \leftarrow \psi$ . However, as actual computation will be done in  $R_p$ , we need to compute  $\mathbf{k} \in R_p$ , which satisfies the following.

$$B_{\text{DD}} \cdot \mathbf{k} = \mathbf{u}_{\text{DD}} \mathbf{f}_{\text{DD}} + \mathbf{e}_{\text{DD}} - [\mathbf{u}_{\text{DD}} \mathbf{f}_{\text{DD}} + \mathbf{e}_{\text{DD}}]_{B_{\text{DD}}} \pmod{p}$$

This trick significantly reduces the cost of verifying the bound of  $\mathbf{e}$ . Below, we summarize the modifications in the setup and distributed decryption algorithms.

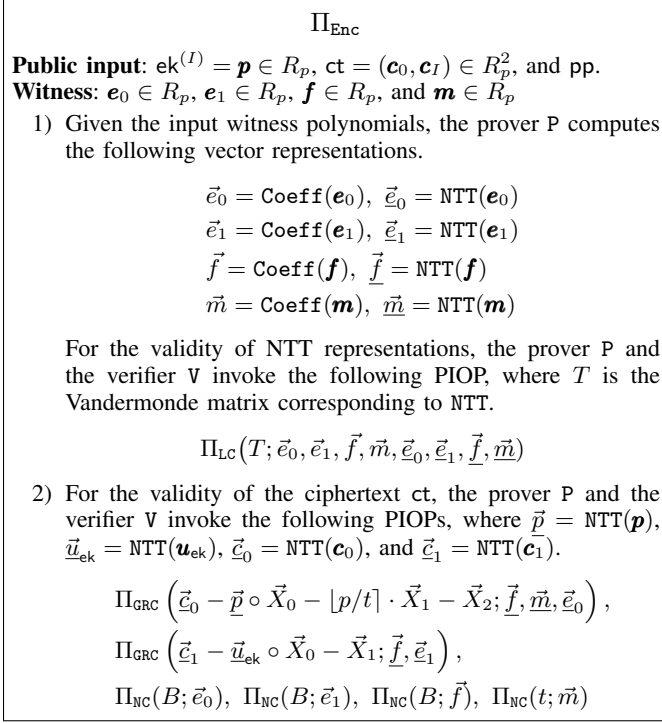


Figure 8. PIOP for BFV.Enc

- **Setup:** Sample a CRS  $\mathbf{u}_{\text{DD}} \leftarrow \mathcal{U}(R_{B_{\text{DD}}})$ .
- **Distributed Decryption:** Sample  $\mathbf{f}_{\text{DD}} \leftarrow \chi$ ,  $\mathbf{e}_{\text{DD}} \leftarrow \psi$ , and return a partial decryption  $\mathbf{d}^{(i)} = \mathbf{c}_l \mathbf{s} + [\mathbf{u}_{\text{DD}} \mathbf{f}_{\text{DD}} + \mathbf{e}_{\text{DD}}]_{B_{\text{DD}}} \pmod{p}$ .

In Fig. 9, we present a PIOP for distributed decryption. The analysis of  $\Pi_{\text{DD}}$  is as follows.

**Theorem 8** (PIOP for MGBFV.DDec).  $\Pi_{\text{DD}}$  is an HVZK PIOP with the following complexity, where  $k_1 = \lfloor \log B \rfloor + 1$  and  $k_3 = \lfloor \log((N+1)B) \rfloor + 1$ .

- The soundness error is  $O\left(\frac{(k_1+k_3)N}{|C|}\right)$
- The prover time is  $O(N(\log N + k_1 + k_3))$
- The verifier time is  $O(N \log N + k_1 + k_3)$
- The query complexity is  $8k_1 + 2k_3 + 20$  and the total number of distinct query points is 1.
- The size of proof oracles is  $(4k_1 + k_3)(4L - N - 2) + 21L - 5N - 5$ . Additionally, P sends one field elements to verifier.
- The size of witness is  $4N$ , 4 elements in  $R_p$ .

## 5. Compilation to zk-SNARK

In this section, we compile our PIOP into a zk-SNARK using a polynomial commitment scheme (PCS). First, we demonstrate how we instantiate our PIOP, including the choice of PCS and the parameter setting for MGHE. Then, we present experimental results for the compiled zk-SNARK, based on concrete proof sizes and runtimes for both the prover and verifier.

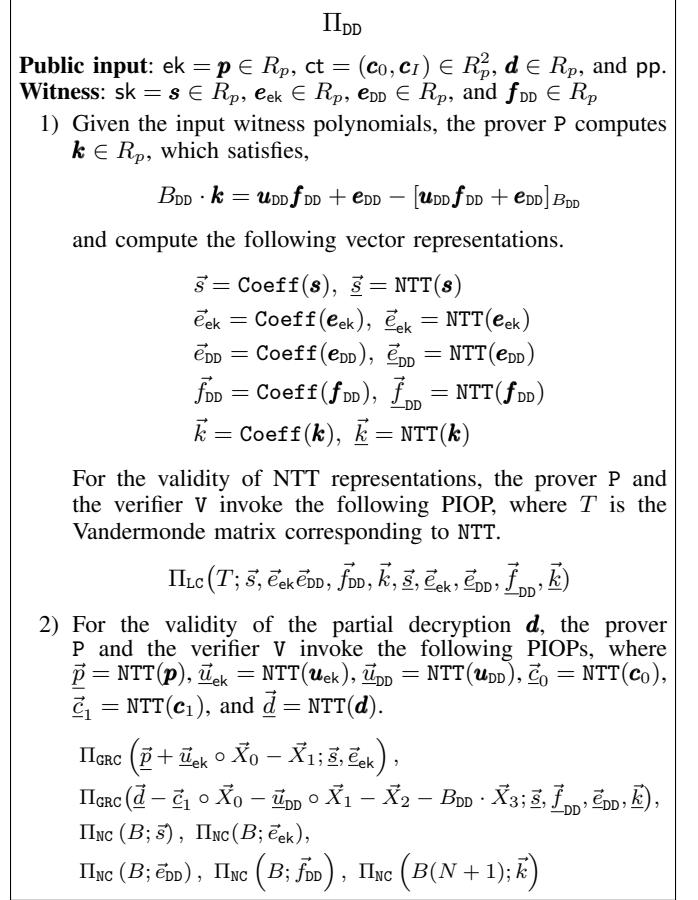


Figure 9. PIOP for MGBFV.DDec

## 5.1. Instantiation

**Polynomial Commitment.** To compile our PIOP into a zk-SNARK, we need a PCS that supports the hiding property and is compatible with NTT-friendly prime fields. Among the possible candidates, we use the lattice-based PCS by Hwang et al. [29], which we refer to as the HSS scheme, since it provides fast proving performance and efficiently supports large prime fields. Additionally, it offers a transparent setup and post-quantum security. We note that other PCS can also be used, thanks to the modularity of PIOP.

In the HSS scheme, the polynomial modulus  $p$  is of the form  $b^r + 1$ , where  $r$  is a power of two. The performance of the scheme depends only on the size of  $b$ , rather than  $p$ , as the space of commitment is determined by  $b$ . Thus, if we choose a proper even number  $b$  so that  $p = b^r + 1$  becomes a prime such that  $p \approx q$ , then it automatically becomes NTT-friendly as  $2^r \mid p - 1$ . While it offers fast prover performance, it results in a larger proof size, as the complexity is proportional to the square root of the input witness size. However, in our instantiation, the proof size usually remains smaller compared to the MGHE public key size.

**Complexity.** As described in Theorem 1, when compiling PIOP into zk-SNARK, additional overheads from the PCS are added to the complexity of the prover and verifier, as well as the proof size. In the HSS scheme, the prover complexity is quasi-linear in witness size, while the verifier complexity and the proof size are square-root in witness size. In Theorems 6 to 8, the prover and verifier complexity is already quasi-linear in witness size, so its asymptotic complexity remains unchanged after compilation. For the proof size, the number of query points is constant, so it results in a square-root proof size in the size of the proof oracles.

**Parameters.** For the parameters for MGHE, we use the uniform ternary distribution  $U(R_3)$  for both the key distribution  $\chi$  and the error distribution  $\psi$ , to attain the minimum bound  $B = 1$ . This choice of distributions is commonly utilized in zero-knowledge proof systems for RLWE samples [39, 40, 49, 50] to reduce the cost of proof generation, but it can be easily generalized to larger values of  $B$ , as the overhead grows with respect to  $\log B$ . For the ring dimension  $N$ , we use  $2^{14}$  and  $2^{15}$ , which provide sufficiently large ciphertext moduli for general homomorphic evaluations, including BFV bootstrapping [51, 52, 53]. For the ciphertext modulus  $q$ , we choose the largest value for each ring dimension, which provides 128-bit security when estimated from the lattice estimator by Albrecht et al. [54]. For the RNS primes  $q_i$ 's, 50–60 bit-sized primes are typically used. We use 53–54 bit primes to provide levels  $\ell = 8$  and  $\ell = 16$  for ring dimensions  $N = 2^{14}$  and  $N = 2^{15}$ , respectively, with the gadget dimension  $\delta$  fixed at 4. For the coefficient modulus  $p = b^r + 1$ , we fix  $r = 32$  and  $64$  for  $N = 2^{14}$  and  $2^{15}$ , respectively, and find the proper value of  $b$  that makes  $p$  prime and  $p \approx q$ . For the parameters  $L$  and  $C$  for PIOP, we set  $L = 2N$  and  $|C| \approx 2^{128}$ . In Table 1, we summarize the parameter setting.

	$N$	$\lceil \log q \rceil$	$\lceil \log p \rceil$	$\ell$	$\delta$	$b$	$r$
Params I	$2^{14}$	429	429	8	4	10792	32
Params II	$2^{15}$	865	865	16	4	11710	64

TABLE 1. PARAMETER SETS

## 5.2. Evaluation

To present the concrete efficiency of our PIOP, we implement  $\Pi_{\text{KG}}$  at a proof-of-concept level using the HSS polynomial commitment scheme. The parameter for the HSS scheme is set to provide the smallest proof size while ensuring 128-bit security. The detailed parameter sets are deferred to Appendix B. All experiments were performed with a single thread on a machine with an Intel(R) Xeon(R) Platinum 8268 CPU running at 2.90GHz and 384GB of RAM. Our source code is available at: <https://github.com/SNUCP/buckler>, which is based on the implementation of the HSS scheme provided in [29].

In Table 2, we provide benchmark results for each parameter set in Table 1. We consider two cases:  $|\Phi| = 0$

	Prover (s)	Verifier (s)	Proof (MB)	Key (MB)
Params I ( $ \Phi  = 0$ )	253	27.8	21.9	18.4
Params I ( $ \Phi  = 2$ )	394	38.3	28.4	31.8
Params II ( $ \Phi  = 0$ )	756	75.7	46.7	74.3
Params II ( $ \Phi  = 2$ )	1280	109	61.2	128

TABLE 2. BENCHMARK RESULTS OF  $\Pi_{\text{KG}}$

and  $|\Phi| = 2$ . For the first case, only the encryption key and relinearization keys are generated, covering the scenario of private set intersection [3] or setup for SPDZ [36]. For the second case, two automorphism keys are additionally generated, corresponding to  $X \mapsto X^5$  and  $X \mapsto X^{-1}$ , the generators of  $\text{Aut}(R)$ . Using these automorphism keys, an evaluator generates other automorphism keys following the method in [55].

For the proof size, it remains similar to the key size when  $N = 2^{14}$ , but becomes about 2 times smaller than the key size in the case of  $N = 2^{15}$ . This is due to the square root complexity of the HSS scheme in proof size. We expect the gap between key and proof size to grow even larger for larger parameters, such as  $N = 2^{16}$ , due to the square root complexity. For the proof generation, it takes about 13 minutes for  $N = 2^{15}$  without automorphism keys. We note that this provides deployable performance for maliciously secure setup for SPDZ, as the previous MPC-based solution [37] takes about 155 minutes to generate public keys for similar parameters.

We also compare performance with the previous state-of-the-art construction by Chatel et al. [23] in Table 3. Although it provides a proof system for relinearization keys and automorphism keys, it only benchmarks proof generation for the encryption key. Thus, for a fair comparison, we modify  $\Pi_{\text{KG}}$  to prove the well-formedness of only the encryption key. The benchmark results from the previous work are taken from Table 4 in [23]. Since it measures performance for a single subring  $R_{q_i}$ , we multiply each metric by the level parameter  $\ell$ , because it shows linear growth with respect to  $\ell$ , as described in Table 5 in [23]. The benchmark results show that our implementation achieves a 5.5x reduction in proof size, a 70x speed-up in proof generation, and a 343x improvement in verification for  $N = 2^{15}$ , which demonstrates the practicality of our PIOP.

	$N$	Prover (s)	Verifier (s)	Proof (MB)
[23]	$2^{14}$	732	370	25.4
Params I (ek)	$2^{14}$	25.0	2.66	8.17
[23]	$2^{15}$	5710	2613	92.8
Params II (ek)	$2^{15}$	81.2	7.61	17.0

TABLE 3. COMPARISON WITH [23]

## References

- [1] S. Angel, H. Chen, K. Laine, and S. Setty, "Pir with compressed queries and amortized query processing," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 962–979.
- [2] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server pir via fhe composition," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 930–947.
- [3] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1223–1237.
- [4] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled psi from homomorphic encryption with reduced computation and communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1135–1150.
- [5] Z. Liu and E. Tromer, "Oblivious message retrieval," in *Annual International Cryptology Conference*. Springer, 2022, pp. 753–783.
- [6] Z. Liu, E. Tromer, and Y. Wang, "Group oblivious message retrieval," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4367–4385.
- [7] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12403–12422.
- [8] J. H. Ju, J. Park, J. Kim, M. Kang, D. Kim, J. H. Cheon, and J. H. Ahn, "Neujeans: Private neural network inference with joint optimization of convolution and bootstrapping," in *ACM CCS*, 2024.
- [9] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 291–311, 2021.
- [10] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song, "Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 726–740.
- [11] H. Kwak, D. Lee, Y. Song, and S. Wagh, "A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation," in *International Conference on Applied Cryptography and Network Security*. Springer, 2024, pp. 403–430.
- [12] Q. Guo, D. Nabokov, E. Suvanto, and T. Johansson, "Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures," in *Usenix Security*, 2024.
- [13] J. H. Cheon, H. Choe, A. Passelègue, D. Stehlé, and E. Suvanto, "Attacks against the indcpa-d security of exact fhe schemes," in *ACM CCS*, 2024.
- [14] M. Checri, R. Sirdey, A. Boudguiga, and J.-P. Bultel, "On the practical cpa d security of "exact" and threshold fhe schemes and libraries," in *Annual International Cryptology Conference*. Springer, 2024, pp. 3–33.
- [15] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012, pp. 1219–1234.
- [16] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 483–501.
- [17] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key fhe," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 735–763.
- [18] R. Ostrovsky, A. Paskin-Cherniavsky, and B. Paskin-Cherniavsky, "Maliciously circuit-private fhe," in *Annual Cryptology Conference*. Springer, 2014, pp. 536–553.
- [19] W. Chongchitmate and R. Ostrovsky, "Circuit-private multi-key fhe," in *IACR International Workshop on Public Key Cryptography*. Springer, 2017, pp. 241–270.
- [20] R. Del Pino, V. Lyubashevsky, and G. Seiler, "Short discrete log proofs for fhe and ring-lwe ciphertexts," in *IACR International Workshop on Public Key Cryptography*. Springer, 2019, pp. 344–373.
- [21] C. Boschini, J. Camenisch, M. Ovsiankin, and N. Spooner, "Efficient post-quantum snarks for rsis and rlwe and their applications to privacy," in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 2020, pp. 247–267.
- [22] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez, "Rinocchio: Snarks for ring arithmetic," *Journal of Cryptology*, vol. 36, no. 4, p. 41, 2023.
- [23] S. Chatel, C. Mouchet, A. U. Sahin, A. Pyrgelis, C. Troncoso, and J.-P. Hubaux, "Pelta-shielding multiparty-fhe against malicious adversaries," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 711–725.
- [24] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zk snarks with universal and updatable srs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2020, pp. 738–768.
- [25] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2020, pp. 677–706.
- [26] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
- [27] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.
- [28] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for r1cs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 103–128.
- [29] I. Hwang, J. Seo, and Y. Song, "Concretely efficient lattice-based polynomial commitment from standard assumptions," in *Annual International Cryptology Conference*. Springer, 2024, pp. 414–448.
- [30] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 2010, pp. 177–194.
- [31] S. Kim, H. Lee, and J. H. Seo, "Efficient zero-knowledge arguments in discrete logarithm setting: sublogarithmic proof or sublinear verifier," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 403–433.
- [32] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," in *Theory of Cryptography Conference*. Springer, 2021, pp. 1–34.
- [33] T. Xie, Y. Zhang, and D. Song, "Orion: Zero knowledge proof with linear prover time," in *Annual International Cryptology Conference*. Springer, 2022, pp. 299–328.
- [34] L. de Castro, A. Polychroniadou, and D. Escudero, "Privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption," in *Neurips Safe Generative AI Workshop 2024*.
- [35] W. Cho, G. Hanrot, T. Kim, M. Park, and D. Stehlé, "Fast and accurate homomorphic softmax evaluation," in *ACM CCS*, 2024.

- [36] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.
- [37] D. Rotaru, N. P. Smart, T. Tanguy, F. Vercauteren, and T. Wood, “Actively secure setup for spdz,” *Journal of Cryptology*, vol. 35, no. 1, p. 5, 2022.
- [38] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.
- [39] T. Attema, V. Lyubashevsky, and G. Seiler, “Practical product proofs for lattice commitments,” in *Annual International Cryptology Conference*. Springer, 2020, pp. 470–499.
- [40] M. F. Esgin, N. K. Nguyen, and G. Seiler, “Practical exact proofs from lattices: New techniques to exploit fully-splitting rings,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 259–288.
- [41] V. Lyubashevsky, N. K. Nguyen, and G. Seiler, “Practical lattice-based zero-knowledge proofs for integer relations,” in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1051–1070.
- [42] A. Bois, I. Cascudo, D. Fiore, and D. Kim, “Flexible and efficient verifiable computation on encrypted data,” in *IACR International Conference on Public-Key Cryptography*. Springer, 2021, pp. 528–558.
- [43] D. Fiore, A. Nitulescu, and D. Pointcheval, “Boosting verifiable computation on encrypted data,” in *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II 23*. Springer, 2020, pp. 124–154.
- [44] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.
- [45] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, “Hyperplonk: Plonk with linear-time prover and high-degree custom gates,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 499–530.
- [46] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Annual cryptology conference*. Springer, 2012, pp. 868–886.
- [47] S. Halevi, Y. Polyakov, and V. Shoup, “An improved rns variant of the bfv homomorphic encryption scheme,” in *Cryptographers’ Track at the RSA Conference*. Springer, 2019, pp. 83–105.
- [48] S. Ling, K. Nguyen, D. Stehlé, and H. Wang, “Improved zero-knowledge proofs of knowledge for the isis problem, and applications,” in *International workshop on public key cryptography*. Springer, 2013, pp. 107–124.
- [49] J. Bootle, V. Lyubashevsky, N. K. Nguyen, and G. Seiler, “More efficient amortization of exact zero-knowledge proofs for lwe,” in *European Symposium on Research in Computer Security*. Springer, 2021, pp. 608–627.
- [50] J. Bootle, V. Lyubashevsky, and G. Seiler, “Algebraic techniques for short (er) exact lattice-based zero-knowledge proofs,” in *Annual International Cryptology Conference*. Springer, 2019, pp. 176–202.
- [51] J. Kim, J. Seo, and Y. Song, “Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from cks,” in *ACM CCS*, 2024.
- [52] Z. Liu and Y. Wang, “Relaxed functional bootstrapping: A new perspective on bgv/bfv bootstrapping,” in *ASIACRYPT*, 2024.
- [53] R. Geelen, I. Iliashenko, J. Kang, and F. Vercauteren, “On polynomial functions modulo  $pe$  and faster bootstrapping for homomorphic encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 257–286.
- [54] M. R. Albrecht, R. Player, and S. Scott, “On the concrete hardness of learning with errors,” *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.
- [55] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No, “Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 36–68.

## Appendix A. Deferred Proofs

### A.1. Proof sketch of Theorem 2 and Theorem 3

The main idea behind the two PIOPs,  $\Pi_{RC}$  and  $\Pi_{LC}$ , is to convert algebraic constraints into polynomial equations, such as  $\mathbf{f}(x) = 0$  for all  $x \in H$ , which is equivalent to  $\mathbf{f}$  being divisible by the vanishing polynomial  $\mathbf{z}_H$ . To demonstrate divisibility, the prover  $P$  sends the quotient polynomial  $\mathbf{q}$ , and the verifier  $V$  checks the polynomial equation  $\mathbf{f}(X) = \mathbf{q}(X) \cdot \mathbf{z}_H(X)$  at a randomly chosen point. For the algebraic-to-polynomial conversion to be correct, the following condition must hold.

Regarding soundness, if the algebraic constraints are unsatisfiable, the resulting polynomial equation will also be unsatisfiable. For the protocol to be accepted,  $P$  must provide a polynomial oracle  $\llbracket \mathbf{q}' \rrbracket$  that satisfies  $\mathbf{f}(\alpha) = \mathbf{q}'(\alpha) \cdot \mathbf{z}_H(\alpha)$ , even if  $\mathbf{f}(X) \neq \mathbf{q}'(X) \cdot \mathbf{z}_H(X)$ , where  $\alpha$  is a query point chosen by  $V$ . The probability of this succeeding is at most  $O(\deg(\mathbf{f})/|C|)$ , which is negligible when  $|C|$  is exponentially large. Thus, soundness is achieved. Furthermore, because a sound PIOP satisfies knowledge soundness (Lemma 2.3, [45]), we can conclude that both  $\Pi_{RC}$  and  $\Pi_{LC}$  provide knowledge soundness.

To ensure HVZK, the prover  $P$  applies randomized encoding to the witness vectors. Due to bounded independence, the distributions of the polynomial oracle responses from the randomized encoded polynomial  $\llbracket \text{REcd}(\vec{a}) \rrbracket$  and from a randomly sampled polynomial  $\llbracket \mathbf{p} \rrbracket$  are indistinguishable for up to  $L - N$  queries. Since the query complexity of PIOPs  $\Pi_{RC}$  and  $\Pi_{LC}$  is 1, the PIOPs can be simulated by substituting randomized encoding with polynomial sampling over  $\mathbb{Z}_p^{<L}[X]$ .

For further details, please refer to [28].

### A.2. Analysis of $\Pi_{GRC}$ in Fig. 5 (Proof of Theorem 4)

The proof of Theorem 4 follows a similar approach to Theorem 2. The primary difference between  $\Pi_{RC}$  and  $\Pi_{GRC}$  lies in the encoding process of the  $k$ -ary polynomial  $\vec{C}$ . In this case, the degree of the quotient polynomial  $\mathbf{q}$  increases from  $d(L - 1) - N$  to  $d(L - 1) - 1$ , due to the coefficient polynomials of  $\mathbf{C}$ , which have degree at most  $N - 1$ . However, the main framework of the PIOP remains the same as in  $\Pi_{RC}$ , allowing us to conclude that  $\Pi_{GRC}$  satisfies completeness, knowledge soundness, and HVZK. Next, we consider the complexity analysis of  $\Pi_{GRC}$ .

- **Soundness Error:**

Instead of checking the entire polynomials  $\hat{\mathbf{a}}_i, \mathbf{q}$ , the verifier accesses polynomial oracles and verifies evaluations returned by these oracles. By the Schwartz-Zippel lemma, the soundness error incurred for each oracle access  $\llbracket \mathbf{f} \rrbracket$  is  $\deg(\mathbf{f})/|C|$ . The sum of degrees of the polynomial accessed by V is  $O(kN + dN)$ , so the total soundness error of  $\Pi_{\text{GRC}}$  is  $O\left(\frac{(k+d)N}{|C|}\right)$ .

- **Prover time:**

The prover's actions consist of the following three steps.

- 1) Encoding vector of  $k$ -ary polynomials  $\vec{\mathbf{C}} \rightarrow \mathbf{C}$ :  
 $\vec{\mathbf{C}}$  consists of  $U$  coefficient vectors, each of length  $N$  and the encoding cost is equivalent to the sum of encoding cost for each coefficient vectors. Then, total encoding cost is  $O(UN)$
- 2) Randomize Encoding  $k$  polynomials  $\vec{a}_i \rightarrow \hat{\mathbf{a}}$ :  
The total cost for randomized encoding is  $O(kL)$
- 3) Construct  $\mathbf{q} := \mathbf{C}(\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_{k-1})/\mathbf{z}_H$ :  
To construct  $\mathbf{q}$ , the prover P first forms the univariate polynomial  $\mathbf{C}(\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_{k-1})$ . This involves up to  $U$  multiplications of  $d + 1$  polynomials of degree  $< L$ , which can be done in  $O(UdL \log(dL) \log d)$  time using the FFT. Then, the prover divides this univariate polynomial by  $\mathbf{z}_H = X^N - 1$ , with a cost of  $O(dL)$ . Thus, the total cost for constructing  $\mathbf{q}$  is  $O(UdL \log(dL) \log d)$

Total prover time is  $O(UL(d+1) \log(Ld+L) \log(d+1))$ .

- **Verifier time:**

The verifier's action consists of the following three steps.

- 1) Encoding vector of  $k$ -ary polynomials  $\vec{\mathbf{C}} \rightarrow \mathbf{C}$ :  
 $\vec{\mathbf{C}}$  consists of  $U$  coefficient vectors, each of length  $N$ . The encoding cost is equivalent to the sum of encoding costs for these coefficient vectors, resulting in a total encoding cost of  $O(UN)$ .
- 2)  $k$  queries to oracles  $\llbracket \hat{\mathbf{a}}_i \rrbracket, \llbracket \mathbf{q} \rrbracket$  at a random point  $\alpha$ :  
The cost for performing these  $k$  oracle queries at point  $\alpha$  is  $O(k)$ .
- 3) Equation check:  
To verify the equation, V evaluates the polynomial  $\mathbf{C}(\alpha)$  at the point  $(\hat{\mathbf{a}}_0(\alpha), \dots, \hat{\mathbf{a}}_{k-1}(\alpha))$ . This polynomial evaluation requires  $O(Ud)$  field operations. Additionally, V evaluates  $\mathbf{z}_H(\alpha)$  locally, which has a cost of  $O(\log N)$ . Therefore, the total cost for checking the equation is  $O(Ud + \log N)$

Total verifier time is  $O(UN + Ud)$ .

- **Query Complexity and Sizes:**

- The query complexity is  $k + 1$ . The total number of distinct query points is 1.
- The size of proof oracles is  $kL + d(L - 1)$ :  
 $k$  polynomials  $\hat{\mathbf{a}}_0, \dots, \hat{\mathbf{a}}_{k-1}$  of degree  $L - 1$  and one polynomial  $\mathbf{q}$  of degree  $d(L - 1) - 1$
- The size of witness is  $kN$ :  
 $k$  vectors  $\vec{a}_0, \dots, \vec{a}_{k-1}$  of length  $N$ .

### A.3. Analysis of $\Pi_{\text{NC}}$ in Fig. 6 (Proof of Theorem 5)

Since the norm check PIOP  $\Pi_{\text{NC}}$  consists of  $k + 1$  rowcheck PIOPs  $\Pi_{\text{RC}}$ , the complexity and security properties of  $\Pi_{\text{NC}}$  follow from those of  $\Pi_{\text{RC}}$ .

By the completeness, knowledge soundness, and HVZK of  $\Pi_{\text{RC}}$  in Theorem 2,  $\Pi_{\text{NC}}$  achieves perfect completeness, knowledge soundness, and HVZK.

Next, We consider the complexity analysis of the norm check PIOP  $\Pi_{\text{NC}}$ .

- **Soundness Error:**

The soundness error from each invocation of  $\Pi_{\text{RC}}(X_0^3 - X_0; \vec{a}_i)$  is  $O\left(\frac{L}{|C|}\right)$  for each  $i = 0, \dots, k - 1$ . The soundness error from the final invocation of  $\Pi_{\text{RC}}$  is  $O\left(\frac{kL}{|C|}\right)$ . Thus, the total soundness error is  $O\left(\frac{kL}{|C|}\right) = O\left(\frac{kN}{|C|}\right)$ , where  $L = O(N)$ .

- **Prover time:**

The prover's actions consist of the following four steps.

- 1) Decompose norm bound,  $B \rightarrow (B_0, \dots, B_{k-1})$ :  
The decomposition consists of  $O(k)$  field operations.
- 2) Decompose a witness vector,  $\vec{a} \rightarrow (\vec{a}_0, \dots, \vec{a}_{k-1})$ :  
The decomposition cost is  $O(kN)$ .
- 3)  $k$  times  $\Pi_{\text{RC}}$  with  $d = 3, U = 1$  polynomial:  
Following the prover time of  $\Pi_{\text{RC}}$  (Theorem 2), the total prover cost is  $O(kN \log N)$ , where  $L = O(N)$ .
- 4) 1 time  $\Pi_{\text{RC}}$  with  $d = 1, U = k + 1$  polynomial:  
Following the prover time of  $\Pi_{\text{RC}}$ , the prover cost is  $O(kN)$ .

The total prover time is  $O(kN \log N)$

- **Verifier time:**

The verifier's actions consist of the following three steps.

- 1) Decompose norm bound,  $B \rightarrow (B_0, \dots, B_{k-1})$ :  $O(k)$
- 2)  $k$  times  $\Pi_{\text{RC}}$  with  $d = 3, U = 1$  polynomial:  
The dominant term in the verifier time for each  $\Pi_{\text{RC}}$  PIOP is  $O(\log N)$ , which is the cost of computing  $\mathbf{z}_H(\alpha)$ . Since the  $\Pi_{\text{NC}}$  verifier checks multiple polynomial equations at a single point, V only needs to compute  $\mathbf{z}_H(\alpha)$  once. Therefore, the total cost is  $O(k + \log N)$ .
- 3) 1 time  $\Pi_{\text{RC}}$  with  $d = 1, U = k + 1$  polynomial:  
Following the verifier time of  $\Pi_{\text{RC}}$  (Theorem 2), the verifier cost is  $O(k + \log N)$ .

The total verifier time is  $O(k + \log N)$ .

- **Query Complexity and Sizes:**

The number of polynomial oracles that the prover sends are described as follows:

- 1) # of deg.  $L - 1$  polys,  $(\hat{\mathbf{a}}_i)_{i=0}^{k-1}$  and  $\hat{\mathbf{a}}$ :  $k + 1$
  - 2) # of deg.  $3(L - 1) - N$  polys,  $(\mathbf{q}_i)_{i=0}^{k-1}$ :  $k$
  - 3) # of deg.  $L - N - 1, \mathbf{q}$ : 1
- The query complexity is  $2k + 2$ .  $2k$  comes from  $k$  times  $\Pi_{\text{RC}}$  with degree 3, and additional 2 comes from the last  $\Pi_{\text{RC}}$  with degree 1. The total number of distinct query points is 1.
  - The size of proof oracles is  $(k + 1)L + k(3(L - 1) - N + 1) + L - N = 4kL - kN + 2L - N - 2k$ .



- The size of witness is  $N$ , a vector  $\vec{a}$  of length  $N$ .

#### A.4. Analysis of $\Pi_{\text{KG}}$ in Fig. 7 (Proof of Theorem 6)

Let us consider the structure of  $\Pi_{\text{KG}}$ . The PIOP  $\Pi_{\text{KG}}$  is reduced to multiple PIOPs as follows:  $(|\Phi| + 3)\delta + 3$  norm check PIOP  $\Pi_{\text{NC}}$  for the length  $N$  vector, 1 linear check PIOP  $\Pi_{\text{LC}}$  for  $4\delta + 3$  pairs of vectors related to the NTT matrix  $T$ ,  $|\Phi|$  linear check PIOP  $\Pi_{\text{LC}}$  for the permutation matrix  $P_\varphi$  for each  $\varphi \in \Phi$ , and  $(|\Phi| + 3)\delta + 1$  general row check PIOP  $\Pi_{\text{GRC}}$  for degree-1 polynomials with at most 4 nonzero entries.

By the completeness, knowledge soundness, and HVZK properties of  $\Pi_{\text{LC}}$  (Theorem 3),  $\Pi_{\text{GRC}}$  (Theorem 4), and  $\Pi_{\text{NC}}$  (Theorem 5), the PIOP  $\Pi_{\text{KG}}$  in Fig. 7 achieves perfect completeness, knowledge soundness, and HVZK.

We consider the complexity analysis of the PIOP  $\Pi_{\text{KG}}$ .

##### • Soundness Error:

The soundness error from invocation of  $O(|\Phi|\delta)$  times  $\Pi_{\text{NC}}$  is  $O\left(\frac{|\Phi|\delta k N}{|C|}\right)$ , from the  $O(\delta)$  times  $\Pi_{\text{LC}}$  is  $O\left(\frac{(\delta+|\Phi|)kN}{|C|}\right)$ , from the  $O(|\Phi|\delta)$  times  $\Pi_{\text{GRC}}$  is  $O\left(\frac{\delta N}{|C|}\right)$ . Therefore, the total soundness error is  $O\left(\frac{|\Phi|k\delta L}{|C|}\right)$ .

##### • Prover Time:

The prover's actions consist of the following five steps.

- 1) Compute vector representations from **Coeff** and NTT: Before the PIOP reduction, the prover compute vector representation from **Coeff** and NTT. The main overhead is to run  $O(|\Phi|\delta)$  times NTT for  $N - 1$  degree polynomial. Thus, the total cost is  $O(|\Phi|\delta N \log N)$ .
- 2)  $(|\Phi| + 3)\delta + 3$  times  $\Pi_{\text{NC}}$ : Following the  $\Pi_{\text{NC}}$  prover time (Theorem 5), the total cost is  $O(|\Phi|\delta k N \log N)$ , where  $k = \lfloor \log B \rfloor + 1$
- 3) 1 time  $\Pi_{\text{LC}}$  for  $(|\Phi| + 3)\delta + 3$  pairs with NTT mat.  $T$ : Since the NTT matrix  $T$  has  $N^2$  non-zero entries, the prover cost is roughly  $O(N^2 + N \log N + |\Phi|\delta N)$  following Theorem 3. The  $N^2$  complexity is for the matrix multiplication between  $T$  and randomized vector  $\vec{v}$  but this multiplication can be computed at  $O(N \log N)$  field operations, so  $N^2$  factor can be substituted as  $N \log N$ . Therefore, total cost can be written as  $O(N \log N + |\Phi|\delta N)$ .
- 4)  $|\Phi|$  times  $\Pi_{\text{LC}}$  for permutation matrix  $P_\varphi$  for  $\varphi \in \Phi$ : Since each permutation matrix  $P_\varphi$  has  $N$  non-zero entries, the prover cost for each  $\Pi_{\text{LC}}$  with matrix  $P_\varphi$  is  $O(N \log N)$  by Theorem 3. Therefore, the total prover cost is  $O(|\Phi|N \log N)$ .
- 5)  $(|\Phi| + 3)\delta + 1$  times  $\Pi_{\text{GRC}}$  with  $d = 1, U \leq 4$ : By Theorem 4, the prover cost for each  $\Pi_{\text{GRC}}$  is  $O(N \log N)$ . Thus, the total cost is  $O(|\Phi|\delta N \log N)$ .

The total prover time is  $O(|\Phi|\delta N (\log N + k))$ .

##### • Verifier Time:

The verifier's actions consist of the following five steps.

- 1) Compute vector representations from NTT: In the same way in prover, the cost is  $O(|\Phi|\delta N \log N)$
- 2)  $(|\Phi| + 3)\delta + 3$  times  $\Pi_{\text{NC}}$ : For each  $\Pi_{\text{NC}}$ , the verifier time is  $O(k + \log N)$  by

Theorem 5. However, the  $O(\log N)$  is for computing  $\mathbf{z}_H(\alpha)$  for some random  $\alpha$ . Since all  $\Pi_{\text{NC}}$  apply common  $\mathbf{z}_H(\alpha)$ ,  $\mathbf{V}$  do not need compute  $\mathbf{z}_H(\alpha)$  for each  $\Pi_{\text{NC}}$ . Thus, total complexity is  $O(|\Phi|\delta k + \log N)$

- 3) 1 time  $\Pi_{\text{LC}}$  for  $(|\Phi| + 3)\delta + 3$  pairs with NTT mat.  $T$ : In the same reason in prover case, the factor  $K = N^2$  can be substituted to  $N \log N$ . By Theorem 3, total cost can be written as  $O(N \log N + |\Phi|\delta)$ .
- 4)  $|\Phi|$  times  $\Pi_{\text{LC}}$  for permutation matrix  $P_\varphi$  for  $\varphi \in \Phi$ : Since each permutation matrix  $P_\varphi$  has  $N$  non-zero entries, the one  $\Pi_{\text{LC}}$  for  $P_\varphi$  is  $O(N)$  by Theorem 3. Therefore, total cost is  $O(|\Phi|N)$ .
- 5)  $(|\Phi| + 3)\delta + 1$  time  $\Pi_{\text{GRC}}$  with  $d = 1, U \leq 4$ : By Theorem 4, the total cost is  $O(|\Phi|\delta N)$ .

The total verifier time is  $O(|\Phi|\delta (N \log N + k))$ .

##### • Query Complexity and Sizes:

To compute query complexity and size of proof oracles, we count all polynomial oracles sending from  $\mathbf{P}$ . Since some witnesses are commonly used in multiple PIOPs, we first count witnesses, that are randomized encoded and then sent to  $\mathbf{V}$  by  $\mathbf{P}$ , and then count additional polynomials per each PIOP to avoid duplication.

- 1)  $2(|\Phi| + 3)\delta + 6 + |\Phi|$  witnesses:
  - # of deg.  $L - 1$  polys:  $2(|\Phi| + 3)\delta + 6 + |\Phi|$
- 2)  $(|\Phi| + 3)\delta + 3$  times  $\Pi_{\text{NC}}$ :
  - # of deg.  $L - 1$  polys,  $(\hat{\mathbf{a}}_i)_{i=0}^{k-1}$ :  $(|\Phi| + 3)\delta k + 3k$
  - # of deg.  $3(L - 1) - N$  polys,  $(\mathbf{q}_i)_{i=0}^{k-1}$ :  $(|\Phi| + 3)\delta + 3$
  - # of deg.  $L - N - 1$ ,  $\mathbf{q}$ :  $(|\Phi| + 3)\delta + 3$
- 3) 1 time  $\Pi_{\text{LC}}$  for  $(|\Phi| + 3)\delta + 3$  pairs with NTT mat.  $T$ :
  - # of deg.  $L + N - 2$ ,  $\mathbf{g}$ : 1
  - # of deg.  $N - 2$ ,  $\mathbf{r}$ : 1
  - # of deg.  $L - 2$ ,  $\mathbf{q}$ : 1
- 4)  $|\Phi|$  times  $\Pi_{\text{LC}}$  for permutation matrix  $P_\varphi$  for  $\varphi \in \Phi$ :
  - # of deg.  $L + N - 2$ ,  $\mathbf{g}$ :  $|\Phi|$
  - # of deg.  $N - 2$ ,  $\mathbf{r}$ :  $|\Phi|$
  - # of deg.  $L - 2$ ,  $\mathbf{q}$ :  $|\Phi|$
- 5)  $(|\Phi| + 3)\delta + 1$  time  $\Pi_{\text{GRC}}$  with  $d = 1, U \leq 4$ :
  - # of deg.  $L - 2$ ,  $\mathbf{q}$ :  $(|\Phi| + 3)\delta + 1$

- The query complexity is  $(5 + k)(|\Phi| + 3)\delta + 4|\Phi| + 3k + 16$  and the total number of distinct query points is 1.
- The size of proof oracles is  $(kE + 7E + 3|\Phi|)L - 2(E - |\Phi| - 1)N - 3E - 3|\Phi| - 1$ , where  $E = (|\Phi| + 3)\delta + 3$ . Additionally,  $\mathbf{P}$  sends  $1 + |\Phi|$  field elements to  $\mathbf{V}$ , which are induced by  $1 + |\Phi|$  times norm check PIOP  $\Pi_{\text{NC}}$ .
- The size of witness is  $((|\Phi| + 3)\delta + 3)N$ ,  $(3 + |\Phi|)\delta + 3$  elements in  $R_p$ .

#### A.5. Analysis of $\Pi_{\text{Enc}}$ and $\Pi_{\text{DD}}$ (Proof of Theorem 7 and Proof of Theorem 8)

Due to the similar structure of both PIOPs,  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$ , we analyze both PIOPs simultaneously. Let us examine the structure of  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$ . Both are reduced to multiple PIOPs:  $\Pi_{\text{Enc}}$  are reduced 4 norm check PIOP, 1 linear check PIOP, and 2 general row check PIOP, while  $\Pi_{\text{DD}}$  are reduced

5 norm check PIOP, 1 linear check PIOP, and 2 general row check PIOP.

In a similar manner as in Section A.4, the PIOP  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  in Fig. 8 and Fig. 9 achieves perfect completeness, knowledge soundness, and HVZK, respectively.

Now we consider the complexity analysis of both PIOPs. For convenience, we set  $k_1 = \lfloor \log B \rfloor + 1$ ,  $k_2 = \lfloor \log t \rfloor + 1$ ,  $k_3 = \lfloor \log B(N+1) + 1 \rfloor$ .

• **Soundness Error:**

The soundness error is the sum of the errors from the invoked PIOPs.  $\Pi_{\text{Enc}}$  invokes 3 times  $\Pi_{\text{NC}}$  with bound  $B$  and 1 time with bound  $t$ , 1 time  $\Pi_{\text{LC}}$ , and 2 times  $\Pi_{\text{GRC}}$  with constant degree and non-zero entries. On the other hands,  $\Pi_{\text{DD}}$  invokes 4 times  $\Pi_{\text{NC}}$  with bound  $B$ , 1 time with bound  $N(B+1)$ , 1 time  $\Pi_{\text{LC}}$ , and 2 times  $\Pi_{\text{GRC}}$  with constant degree and non-zero entries. By Theorem 5, Theorem 3, and Theorem 4, the soundness errors of  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $O\left(\frac{(k_1+k_2)N}{|C|}\right)$  and  $O\left(\frac{(k_1+k_3)N}{|C|}\right)$ , respectively.

• **Prover Time:**

- 1) Compute vector representations:  
Before the PIOP reduction, the  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  provers run 4 and 5 times `Coeff`, and 8 and 11 times `NTT`, respectively. Then, total cost of both PIOPs can be written as  $O(N \log N)$ .
- 2) 4 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{Enc}}$ ) / 5 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{DD}}$ ):  
By Theorem 5, the prover cost for  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $O((k_1+k_2)N \log N)$  and  $O((k_1+k_3)N \log N)$ , respectively.
- 3) 1 time  $\Pi_{\text{LC}}$  with NTT matrix  $T$ :  
In the similar manner in Section A.4, the overwhelming operation for  $\Pi_{\text{LC}}$  is NTT matrix multiplication. Then, the costs of both PIOPs are  $O(N \log N)$ .
- 4) 2 times  $\Pi_{\text{GRC}}$  with  $d=1, U \leq 5$  polynomials:  
By Theorem 4, the prover costs of both PIOPs are  $O(N \log N)$ .

Therefore, the total prover times of  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $O((k_1+k_2)N \log N)$  and  $O((k_1+k_3)N \log N)$ , respectively.

• **Verifier Time:**

- 1) Compute vector representations from NTT:  
Before the PIOP reduction the  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  verifiers run 4 and 6 times `NTT`, respectively. Thus, the verifier costs of both PIOPs are  $O(N \log N)$ .
- 2) 4 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{Enc}}$ ) / 5 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{DD}}$ ):  
By Theorem 5, the verifier cost for  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $O(k_1+k_2+\log N)$  and  $O(k_1+k_3+\log N)$ , respectively.
- 3) 1 time  $\Pi_{\text{LC}}$  with NTT matrix  $T$ :  
Similarly in the prover case, the costs of both PIOPs are  $O(N \log N)$ .
- 4) 2 times  $\Pi_{\text{GRC}}$  with  $d=1, U \leq 5$  polynomials:  
By Theorem 4, the verifier costs of both PIOPs are  $O(N)$ .

Therefore, the total verifier times of  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $O(N \log N + k_1 + k_2)$ . and  $O(N \log N + k_1 + k_3)$ , respectively.

• **Query Complexity and Sizes:**

To distinguish the complexity of PIOPs  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$ , we denote the costs for  $\Pi_{\text{Enc}}$ ,  $\Pi_{\text{DD}}$ , and both cases as  $(\Pi_{\text{Enc}})$ ,  $(\Pi_{\text{DD}})$ , and  $(\text{Common})$  respectively.

- 1) 8 witnesses ( $\Pi_{\text{Enc}}$ ) / 10 witnesses ( $\Pi_{\text{DD}}$ ):
    - ( $\Pi_{\text{Enc}}$ ) # of deg.  $L-1$  polys: 8
    - ( $\Pi_{\text{DD}}$ ) # of deg.  $L-1$  polys: 10
  - 2) 4 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{Enc}}$ ) / 5 times  $\Pi_{\text{NC}}$  ( $\Pi_{\text{DD}}$ )::
    - ( $\Pi_{\text{Enc}}$ ) # of deg.  $L-1$  polys,  $(\hat{\mathbf{a}}_i)_{i=0}^{k-1}$ :  $3k_1+k_2$
    - ( $\Pi_{\text{Enc}}$ ) # of deg.  $3(L-1)-N$  polys,  $(\mathbf{q}_i)_{i=0}^{k-1}$ :  $3k_1+k_2$
    - ( $\Pi_{\text{DD}}$ ) # of deg.  $L-1$  polys,  $(\hat{\mathbf{a}}_i)_{i=0}^{k-1}$ :  $4k_1+k_3$
    - ( $\Pi_{\text{DD}}$ ) # of deg.  $3(L-1)-N$  polys,  $(\mathbf{q}_i)_{i=0}^{k-1}$ :  $4k_1+k_3$
    - ( $\text{Common}$ ) # of deg.  $L-N-1$ ,  $\mathbf{q}$ : 4
  - 3) 1 time  $\Pi_{\text{LC}}$  with NTT matrix  $T$ :
    - ( $\text{Common}$ ) # of deg.  $L+N-2$ ,  $\mathbf{g}$ : 1
    - ( $\text{Common}$ ) # of deg.  $N-2$ ,  $\mathbf{r}$ : 1
    - ( $\text{Common}$ ) # of deg.  $L-2$ ,  $\mathbf{q}$ : 1
  - 4) 2 times  $\Pi_{\text{GRC}}$  with  $d=1, U \leq 5$ :
    - ( $\text{Common}$ ) # of deg.  $L-2$ ,  $\mathbf{q}$ : 2
- The query complexity of  $(\Pi_{\text{Enc}})$  and  $(\Pi_{\text{DD}})$  are  $6k_1+2k_2+17$  and  $8k_1+2k_3+20$ , respectively. In both cases, The total number of distinct query points is 1.
  - The size of proof oracles of  $(\Pi_{\text{Enc}})$  and  $(\Pi_{\text{DD}})$  are  $(3k_1+k_2)(4L-N-2)+18L-4N-5$  and  $(4k_1+k_3)(4L-N-2)+21L-5N-5$ , respectively. In both cases,  $\mathcal{P}$  sends one field elements to verifier.
  - In both cases, the size of witness is  $4N$ , 4 elements in  $R_p$ . The witness of  $\Pi_{\text{Enc}}$  and  $\Pi_{\text{DD}}$  are  $(\mathbf{e}_0, \mathbf{e}_1, \mathbf{f}, \mathbf{m})$  and  $(\mathbf{s}, \mathbf{e}_{\text{ek}}, \mathbf{e}_{\text{DD}}, \mathbf{f}_{\text{DD}})$ , respectively.

## Appendix B. Parameters for Polynomial Commitment

In the HSS scheme, we configure the parameters  $\log q$ ,  $N$ ,  $n$ ,  $d$ ,  $\mu$ , and  $\nu$  for the evaluation protocol. For detailed definitions, see Table 1 in [29]. The input size  $N$  is calculated from Theorem 6, and all polynomials are evaluated in a batched manner.

	$\lceil \log q \rceil$	$N$	$n$	$d$	$\mu$	$\nu$
Params I (ek)	100	$\approx 26 \cdot 2^{14}$	$2^{12}$	$2^{11}$	1	2
Params I ( $ \Phi =0$ )	100	$\approx 212 \cdot 2^{14}$	$2^{13}$	$2^{11}$	1	2
Params I ( $ \Phi =2$ )	100	$\approx 340 \cdot 2^{14}$	$2^{13}$	$2^{11}$	1	2
Params II (ek)	112	$\approx 26 \cdot 2^{15}$	$2^{12}$	$2^{11}$	1	2
Params II ( $ \Phi =0$ )	112	$\approx 212 \cdot 2^{15}$	$2^{13}$	$2^{11}$	1	2
Params II ( $ \Phi =2$ )	112	$\approx 340 \cdot 2^{15}$	$2^{13}$	$2^{11}$	1	2

TABLE 4. PARAMETERS FOR THE HSS SCHEME