# DekartProof: Efficient Vector Range Proofs and Their Applications

Dan Boneh[1], Trisha Datta[1], Rex Fernando[2], Kamilla Nazirkhanova[1], and Alin Tomescu[2]

[1] Stanford University
[2] Aptos Labs

**Abstract.** Let $p$ be a prime and consider a committed vector $\boldsymbol{v} = (v_1, \ldots, v_m) \in \mathbb{F}_p^m$. We develop new techniques for succinctly proving in zero-knowledge that all the elements of $\boldsymbol{v}$ are in the range $\{0, 1, \ldots, n\}$ for some $n < p$. We refer to this as a batched zero-knowledge range proof, or a batched ZKRP. This problem comes up often in cryptography: it is needed in publicly verifiable secret sharing (PVSS), confidential transactions, and election protocols. Our approach makes use of a multilinear polynomial commitment scheme and the sum check protocol to efficiently provide a batch range proof for the entire vector. Along the way we introduce a new type of a Polynomial Interactive Oracle Proof (PIOP) we call a *Homomorphic PIOP* that can be compiled into a SNARK. We use an HPIOP to construct a new efficient zero-knowledge version of the sum check protocol. We compare our new techniques with existing range proofs and lookup arguments.

**Keywords:** zero-knowledge, range proofs, publicly verifiable secret sharing, confidential transactions

## 1 Introduction

A zero-knowledge range proof (ZKRP) [14] is a protocol for proving in zero knowledge that a committed or encrypted value in $\mathbb{F}_p$ lies in a specific range $[0, n) \subset \mathbb{F}_p$. In this paper we are primarily interested in the *batch* version of this problem, where the goal is to prove that all the elements in a vector are in $[0, n)$. That is, we are interested in a succinct zero-knowledge non-interactive argument for the following instance-witness relation:

$$\left\{ (\mathsf{com}; \boldsymbol{v} \in \mathbb{F}_p^m) \quad : \quad \mathsf{com} \in \mathsf{Commit}(\boldsymbol{v}), \quad \boldsymbol{v} \in [0, n)^m \right\}$$

where $\mathsf{Commit}$ is the commitment algorithm of a hiding and binding commitment scheme. This problem comes up often in cryptography:

- In several publicly verifiable secret sharing (PVSS) protocols for field elements (e.g., [33]), one uses additive ElGamal encryption, where the plaintext is encoded in the exponent of a group element. During decryption, the decryptor must compute a discrete log to recover the plaintext field element. To make sure this is feasible, the encryptor must provide a zero-knowledge proof that all the plaintexts in a vector of ElGamal ciphertexts are sufficiently small. Hence, a fast batch ZKRP enables a fast PVSS. Here we are treating ElGamal encryption as a commitment scheme.
- In election systems that use additively homomorphic encryption (e.g., [16]), a voter casts a ballot containing multiple encrypted votes and must prove that all the ciphertexts on the ballot are encryptions of zero or one.
- In a confidential transactions system for Bitcoin (e.g., [27]) one must prove that all outputs of a transaction are an encryption of a positive value, namely a value in an interval $[0, n)$ for some upper bound $n$. The same holds in other confidential transactions systems (e.g., [5]).

In all these examples, efficient batch ZKRP enables highly efficient constructions. A number of existing works construct efficient batch ZKRP protocols [6,15,21,2]. A batch ZKRP can also be obtained from a general lookup argument such as cq [17], Lasso [32], and others [20,37,28,19,38].

**Our contributions.** We construct a new highly efficient batch ZKRP protocol called DekartProof. The prover work is $O(m \log n)$ to prove that the elements of a vector $\boldsymbol{v}$ of length $m$ are in $[0, n)$, the size of the structured reference string is linear in $m$, and the verifier work and proof size depend only logarithmically on $m$ and $n$. In Section 1.2 we show that this new construction compares favorably in prover and verifier times to existing techniques; see Table 1 for a summary. As in prior work [21], we can generalize DekartProof to prove that a set of values is in arbitrary ranges, that is $v_i \in [\ell_i, h_i)$ for all $i \in [m]$, by invoking the DekartProof prover twice (see Section B.2 for details).

DekartProof relies on a zero-knowledge variant of the sum-check protocol [26]. While prior work [11,36,31,35] has introduced constructions for ZK sum-checks, these constructions either require expensive group work in each round or do not guarantee ZK for every round of the sum-check protocol. We introduce an efficient fully ZK sum-check. To formally describe this construction, we also introduce a new type of information-theoretic proof called a *homomorphic polynomial interactive oracle proof* or HPIOP. We present a compiler that shows how to convert an HPIOP into an interactive argument. In particular, we prove that our compilation preserves round-by-round soundness [8]. While round-by-round (knowledge) soundness is necessary for using the Fiat–Shamir to transform a multiround protocol into a non-interactive protocol, prior works sometimes overlook this step and only prove that compilation preserves overall soundness. We also prove that composing round-by-round knowledge sound protocols preserves round-by-round knowledge soundness. This is needed when using DekartProof in some of the applications outlined above. Our ZK sum-check, the HPIOP model, and the composition theorem may be of independent interest.

### 1.1 Technical Overview

We begin by describing the main ideas that underlie the DekartProof batch ZKRP. We start with Borgeaud's range proof [4] for a single value $z$. Both the prover and verifier start with a commitment to a blinded polynomial whose constant term is $z$. To prove to the verifier that $z \in [0, n)$, Borgeaud's prover uses a homomorphic commitment scheme to commit to $\log n$ blinded univariate polynomials $f_1, \ldots, f_{\log n}$ such that $f_i$ encodes the $i^{th}$ bit of $z$ in its constant term. The prover then uses algebraic techniques to prove that these $f_i$'s encode bits in their constant terms and uses the homomorphism of the commitment scheme to prove that the encoded bits form the bit decomposition of $z$.

As is, Borgeaud's method is not batched and requires the prover to compute expensive FFTs. To address these issues and allow a prover to prove that $m$ values $z_1, \ldots, z_m$ are in a range $[0, n)$, DekartProof switches to a multivariate setting. This avoids the need for FFTs and also presents a natural batching approach. The DekartProof prover and verifier both start with a commitment to a multilinear polynomial $f$, where $f$ evaluates to $z_i$ at the $i^{th}$ point on the boolean hypercube. The DekartProof prover still commits to just $\log n$ blinded (multilinear) polynomials, but instead of $f_i$ encoding the $i^{th}$ bit of a single value, each $f_i$ now encodes the $i^{th}$ bits of all values in the batch at different points in the hypercube. That is, the evaluation of $f_i$ at the $j^{th}$ point on the hypercube is the $i^{th}$ bit of $z_j$. We reserve one point on the hypercube for random blinding values for all $f_i$'s. The requisite algebraic checks in the multivariate setting can be reduced to performing a zero-knowledge zero-check, which checks that a polynomial is 0 at every point on the boolean hypercube. From Spartan [30], we can perform a zero-check using a simple variation on the standard sum-check protocol. This means that in order to describe our batch zero-knowledge range proof, we need a zero-knowledge sum-check as a building block.

There are a number of constructions for a zero-knowledge sum-check in the literature [11,36,31,35], but they either require expensive group work in each round of the protocol or leak an evaluation of the sum-check polynomial. These options are unsuitable for our batch ZKRP: efficiency is a top priority, and we want our construction to be fully zero knowledge. For this reason, we introduce a new efficient fully ZK sum-check protocol.

To understand how our fully ZK sum-check protocol works, we first must understand why the standard sum-check protocol is not ZK. The standard sum-check protocol allows a prover to convince a verifier with oracle access to a $\mu$-variate polynomial $f$ that the sum of evaluations of $f$ over the boolean hypercube is $H$. The protocol consists of $\mu$ rounds where the prover sends a univariate polynomial to the verifier and the verifier responds with a random challenge and a final round where the verifier issues a single query to the

oracle to $f$. The prover's univariate polynomial in each round is derived from $f$ as well as all prior random challenges from the verifier. Each round thus leaks information about $f$. Furthermore, at the end of the protocol, the verifier obtains a single evaluation of $f$ from the oracle. We must address both of these sources of leakage to construct a ZK sum-check.

To address the first source of leakage, we follow previous work from Chiesa et al. [11], which introduced a random masking polynomial sent at the beginning of the protocol to mask all evaluations of $f$ revealed by the univariate polynomials sent during the protocol. While their choice of masking polynomial adds potentially exponential overhead, Xie et al. [36] later proposed a far more efficient construction for this masking polynomial. Unfortunately, the masking polynomial alone does not hide the result of the verifier's final oracle query to $f$, which makes this approach insufficient for our use case. To achieve full zero-knowledge, the DekartProof prover generates the polynomial $\hat{f}$ by blinding $f$ at a single point in the boolean hypercube. Practically, this amounts to the DekartProof prover sending an oracle to a blinding polynomial $\beta$ to the verifier, and $\hat{f} \coloneqq f + \beta$. Then the DekartProof prover and verifier perform essentially the standard sum-check protocol over the blinded $\hat{f}$ with the masking polynomial from Xie et al. In the last step of the sumcheck protocol, the verifier only needs a single evaluation of $\hat{f}$ to complete its checks. Because $\hat{f}$ is blinded at a single point, this single evaluation reveals nothing about $f$.

For the above protocol to succeed, the verifier needs to be able to (1) query for the evaluation of $\hat{f} = f + \beta$ and (2) check that the blinding polynomial $\beta$ is of the correct form. Abstractly, the first condition means that the verifier must be able to query for the evaluation of the sum of two polynomial oracles. For the second condition, we always define our blinding polynomial as a public polynomial multiplied by a random scalar, so the verifier must be able to construct a polynomial and then query a polynomial oracle to see if its underlying polynomial is equal to the constructed polynomial multiplied by a scalar. These two types of queries present a challenge for formally describing our ZK sum-check.

We can usually formally define any protocol that deals with polynomial oracles, like the standard sum-check protocol, as a polynomial interactive oracle proof (PIOP) [12,7]. In PIOPs, prover messages are restricted to $\mu$-variate polynomial oracles over some finite field $\mathbb{F}$ that the verifier can query at any point in $\mathbb{F}^\mu$. Unfortunately, these evaluation queries do not encompass the queries necessary for our ZK sum-check, which means that we cannot formally describe our ZK sum-check as a PIOP.

Thus, in order to describe our ZK sum-check formally, we introduce a new type of information theoretic proof called homomorphic polynomial interactive oracle proofs (HPIOPs). While we introduce this model to capture one specific protocol, we believe that it is general enough to apply to several application areas and is therefore of independent theoretical interest. At a high level, HPIOPs mimic the functionality of PIOPs that are compiled with additively homomorphic polynomial commitments. In other words, while prover messages are still restricted to $\mu$-variate polynomial oracles, an HPIOP verifier can query a linear combination of any of the polynomial oracles it has received at any point in $\mathbb{F}^\mu$. This is a generalization of the first type of necessary query for our ZK sum-check and also encompasses the queries allowed in standard PIOPs. Moreover, an HPIOP verifier can construct a polynomial and query any polynomial oracle it has received to check if its underlying polynomial is a scalar multiple of the constructed polynomial. This is exactly the second type of necessary query for our ZK sum-check.

Just like classic PIOPs, HPIOPs can be compiled by replacing prover messages with homomorphic polynomial commitments. We state and prove a compilation theorem that shows how to use a homomorphic polynomial commitment scheme and an additional simple argument of knowledge to compile an HPIOP into an interactive argument. We prove that the resulting interactive argument of knowledge is round-by-round knowledge sound. From Canetti et al. [8], this round-by-round knowledge soundness property allows us to use Fiat-Shamir [18] to transform the compiled interactive multiround argument into a non-interactive argument.

Recall that in our applications of interest, we wish to prove that several committed values are all in a given range. We can use DekartProof to create such a proof by first defining and committing to a polynomial whose evaluations over the boolean hypercube are equal to the committed values and then producing a consistency proof between the polynomial and the commitment. That is, if the range proof shows that the evaluations of some polynomial $f$ over the boolean hypercube are in $[0, n)$, then the prover just needs to

additionally show that the evaluations of $f$ over the hypercube are equal to the values committed to in the commitment. If we are dealing with ElGamal encryption or Pedersen commitments, then a prover can prove consistency with a compressed $\Sigma$-protocol [1] (see Section D for details). To show round-by-round knowledge soundness of such protocols, we prove that a sequential composition of two round-by-round knowledge sound interactive arguments is also a round-by-round knowledge sound interactive argument.

### 1.2 Related Work

**Range Proofs.** For a comprehensive overview of ZKRPs, we refer the reader to this recent SoK [14]. Here we focus specifically on batched ZKRPs and give cost estimates for prover time, verifier time, and proof size in Table 1.

Bulletproofs [6] are inner product arguments that provide highly efficient ZKRPs with logarithmic size proofs and support batch proof verification and proof aggregation. However, they have linear verifier time whereas DekartProof has logarithmic verifier time. Sharp [15] is a family of short relaxed range proofs that supports batching and achieves better performance than Bulletproofs [15, Table 2] for appropriate parameters. However, it only achieves a relaxed notion of soundness. The soundness can be strengthened but requires using groups of hidden order, which results in much larger proof sizes. MissileProof [21], the current state-of-the-art, achieves constant proof size and constant verifier time but larger prover time and SRS than DekartProof. Notably, their prover is required to perform FFTs, whereas DekartProof's prover does not. To achieve their result, the authors first construct a polynomial IOP (PIOP) and apply the DARK compiler [7] to obtain an interactive argument of knowledge. However, the underlying PIOP is not ZK because it reveals polynomial evaluations, so the compiled argument is also not ZK. We believe it is possible to modify the PIOP to achieve ZK using techniques similar to those described in this work without affecting the asymptotic complexity, but this requires a more careful analysis.

Lookup arguments allow to prove that elements of a committed vector $a$ are contained in a (possibly much larger) committed table $T$. If the table represents all values in a specified range, lookup arguments can be used as a batched range proof. Starting with Caulk [37], there has been a line of work [28,19,38,17] focused on shifting online prover work into a pre-processing phase. However, all these schemes require a structured reference string linear in the size of the table, which limits their applicability in practice (e.g., if we would like to solve discrete log for 40 bits). Lasso [32] is a new family of lookup arguments that avoids committing to the entire table when the table is *structured*, making it a great candidate for batched range proofs. However, in its current form, Lasso is not zero-knowledge and modifying it to achieve zero-knowledge is non-trivial. We provide a more detailed discussion in Section 6.

Lastly, Boneh et al. [2] describe a ZKRP using homomorphic polynomial commitments. It can be batched using our techniques, but this batched construction is asymptotically more expensive than DekartProof for settings of interest.

**Zero-Knowledge Sum-Check Variants.** One relevant line of work is the zero-knowledge sum-check introduced by Wahby et al. [35]. They replace the polynomials sent in each round with linearly homomorphic commitments, which introduces expensive group work for both the prover and verifier. We also note that their definition of a zero-knowledge sum-check is slightly different than ours as they commit to the value of the sum with a hiding commitment so that the verifier never learns the actual sum value.

Another line of work [11,36] introduced the use of random masking polynomials to mask the messages sent by the prover in each sum-check round. Unfortunately, this approach still leaks at most one evaluation of the sum-check polynomial to the verifier whereas our zero-knowledge sum-check construction leaks no

[3] These numbers come from instantiating Lasso with Sona [32]. The quantity $c$ is some constant that parameterizes the construction.

[4] The range proof PIOP from [21] is not zero-knowledge as described, but we believe it can be made zero-knowledge without affecting asymptotic costs.

[5] These numbers come from instantiating DekartProof with Zeromorph + KZG[24].

[6] This from a KZG CRS and can be improved by using a ZK homomorphic multilinear polynomial commitment scheme with transparent setup.

| Scheme | Proof size | $\mathcal{P}$ time | $\mathcal{V}$ time | ZK | SRS |
|---|---|---|---|---|---|
| Lasso [32] [3] | $O(\log m \cdot \log\log m)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}$ | $O(cm)$ $\mathbb{F}$, $O(cm + cn^{1/c})$ $\mathbb{G}$ | $O(\log m \cdot \log\log m)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}$ | ✗ | None |
| cq [17] | $O(1)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}_1$ | $O(m \log m)$ $\mathbb{F}$, $O(m)$ $\mathbb{G}_1$ | $O(1)$ $P$ | ✗ | $O(n)$ $\mathbb{G}_1$, $O(n)$ $\mathbb{G}_2$ |
| Bulletproofs [6] | $O(1)$ $\mathbb{F}$, $O(\log m + \log\log n)$ $\mathbb{G}$ | $O(m \log n)$ $\mathbb{F}$, $O(m \log n)$ $\mathbb{G}$ | $O(m \log n)$ $\mathbb{F}$, $O(m \log n)$ $\mathbb{G}$ | ✓ | None |
| MissileProof [21] | $O(1)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}_1$ | $O(m \log m \cdot \log n \log\log n)$ $\mathbb{F}$, $O(m \log n)$ $\mathbb{G}_1$ | $O(1)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}_1$ $O(1)$ $\mathbb{G}_2, 5P$ | ✓[4] | $O(m \log n)$ $\mathbb{G}_1$ $O(1)$ $\mathbb{G}_2$ |
| DekartProof[5] (this work) | $O(\log m)$ $\mathbb{F}$, $O(\log m + \log n)$ $\mathbb{G}_1$ | $O(m \log n)$ $\mathbb{F}$, $O(m \log n)$ $\mathbb{G}_1$ | $O(\log m + \log n)$ $\mathbb{F}$ $O(\log m + \log n)$ $\mathbb{G}_1$, $O(1)$ $\mathbb{G}_2, 5P$ | ✓ | $O(m)$ $\mathbb{G}_1$, $O(1)$ $\mathbb{G}_2$[6] |

**Table 1.** Costs of proving that $m$ values are in $[0, n)$ for various batch range proofs in groups of known order.

evaluations. Setty and Lee [31] introduced a zero-knowledge sum-check that hides the sum in a commitment as in [35] but uses the masking polynomial technique from Xie et al. [36] to avoid computing expensive commitments. This scheme requires a specialized polynomial commitment scheme (PCS) whose verifier work is $O(\sqrt{2^\mu})$, where $\mu$ is the number of variables in the sum-check polynomial. Modifying their scheme to use a more efficient traditional PCS would leak an evaluation of the sum-check polynomial.

## 2 Preliminaries

Let $[n]$ denote the set $\{1, \ldots, n\}$. Let $[i, j]$ denote the set $\{i, i+1, \ldots, j\}$. Let $[i, j)$ denote the set $\{i, i+1, \ldots, j-1\}$. We denote vectors with bold font (e.g., $\boldsymbol{x}$) and label the elements of a vector $\boldsymbol{x}$ of length $n$ as $x_1, \ldots, x_n$. For any set $S$, let $S[i]$ denote the $i^{th}$ element lexicographically in $S$. Let $(0, 1)$ denote the continuous interval between 0 and 1.

We use $\mathbb{F}$ to denote a finite field of prime order. For $\mu \in \mathbb{N}_{\geq 1}$ and $d \in \mathbb{N}$, let $\mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$ denote the set of $\mu$-variate polynomials with coefficients in $\mathbb{F}$ and of individual degree at most $d$. When the number of variables is clear, we sometimes write $f(\boldsymbol{X})$ instead of $f(X_1, \ldots, X_\mu)$ for brevity. If for all $c \in \mathbb{N}$, $f(\lambda)$ is $o(\lambda^{-c})$, then $f(\lambda)$ is $\mathsf{negl}(\lambda)$ or negligible. The boolean hypercube for any dimension $\mu \in \mathbb{N}$ refers to the set of all points $\boldsymbol{b} \in \{0, 1\}^\mu$

For any $x \in \mathbb{N}$, let $\langle x \rangle$ be the bit decomposition of $x$: $\langle x \rangle \to x_1, \ldots, x_n$ such that $x = \sum_{i \in [n]} x_i \cdot 2^{i-1}$ and $x_i \in \{0, 1\}$ for all $i \in [n]$.

An instance-witness relation is a set of pairs $(\mathbb{x}; \mathbb{w})$. For any instance-witness relation $\mathcal{R}$, $\mathcal{L}(\mathcal{R})$ is the set of all instances $x$ for which there is a witness $w$ that satisfies $\mathcal{R}$. An indexed relation is a set of tuples $(\mathbb{i}, \mathbb{x}; \mathbb{w})$ where $\mathbb{i}$ is the index. For any indexed relation $\mathcal{R}$, $\mathcal{L}(\mathcal{R})$ is the set of all index-instance pairs $(\mathbb{i}, \mathbb{x})$ for which there is a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$.

For three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of the same prime order, we use $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, to denote an efficiently computable non-degenerate bilinear pairing. We denote an oracle to a polynomial $f$ as $\mathcal{O}^f$, and $\mathcal{S}^\mathcal{V}$ signifies that the algorithm $\mathcal{S}$ has oracle access to an oracle for $\mathcal{V}$.

We use the notation $\mathsf{F}(\mathcal{P}(x), \mathcal{V}(x')) \to \mathcal{P}(y), \mathcal{V}(y')$ to denote an interactive protocol between $\mathcal{P}$ and $\mathcal{V}$ where $\mathcal{P}$ starts the protocol with input $x$ and ends additionally knowing $y$ and $\mathcal{V}$ starts with input $x'$ and ends additionally knowing $y'$. If one party, $\mathcal{V}$ for instance, has no input, we write the protocol signature as $\mathsf{F}(\mathcal{P}(x), \mathcal{V}) \to \mathcal{P}(y), \mathcal{V}(y')$ for brevity.

### 2.1 Multilinear Extensions

We define the $\mathsf{eq}$ polynomial for any $\boldsymbol{x} \in \mathbb{F}^\mu$ such that for all points $\boldsymbol{b} \in \{0,1\}^\mu$, $\mathsf{eq}_{\boldsymbol{b}}(\boldsymbol{X})$ evaluates to 1 at $\boldsymbol{b}$ and to 0 at all other points in $\{0,1\}^\mu$:

$$\mathsf{eq}_{\boldsymbol{x}}(X_1, \ldots, X_\mu) = \prod_{i \in [\mu]} (x_i X_i + (1 - x_i)(1 - X_i))$$

We additionally define the $\mathsf{vanish}$ polynomial for any $\boldsymbol{x} \in \mathbb{F}^\mu$ such that for all points $\boldsymbol{b} \in \{0,1\}^\mu$, $\mathsf{vanish}_{\boldsymbol{b}}(\boldsymbol{X})$ evaluates to 0 at $\boldsymbol{b}$ and to 1 at all other points in $\{0,1\}^\mu$ (i.e., it "vanishes" on $\boldsymbol{b}$):

$$\mathsf{vanish}_{\boldsymbol{x}}(X_1, \ldots, X_\mu) = 1 - \mathsf{eq}_{\boldsymbol{x}}(X_1, \ldots, X_\mu)$$

**Lemma 1 ([10]).** *For any multilinear polynomial $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq 1}$, $f(\boldsymbol{X}) = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} f(\boldsymbol{b}) \cdot \mathsf{eq}_{\boldsymbol{b}}(\boldsymbol{X})$.*

### 2.2 Interactive and Non-Interactive Arguments

Here we restate the definitions of interactive arguments from Hyperplonk [10].

**Definition 1 (Interactive Arguments [10]).** *An interactive protocol $\Pi$ for an indexed relation $\mathcal{R} = (\mathbb{i}, \mathbb{x}, \mathbb{w})$ is defined by a tuple of algorithms $(\mathsf{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ and consists of an offline non-interactive preprocessing phase run by an indexer $\mathcal{I}$ and an online interactive phase between a prover $\mathcal{P}$ and verifier $\mathcal{V}$. For an index $\mathbb{i}$, public instance $\mathbb{x}$, prover witness $\mathbb{w}$, and $(\mathsf{pp}, \mathsf{vp})$ output by the deterministic indexer $\mathcal{I}$ on $\mathbb{i}$, let the verifier's output be denoted by the random variable $\langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle$. $\Pi$ can be sound, knowledge-sound, public-coin, honest-verifier zero-knowledge, non-interactive, and succinct. We formally define these properties in Section A.2. We define computational completeness and round-by-round soundness below:*

*Computational Completeness: $\Pi$ is computationally complete if for every PPT $\mathcal{A}$:*

$$\Pr\left[ \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;:\; \begin{array}{l} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

*If $\mathcal{A}$ is unbounded, then we say $\Pi$ is perfectly complete.*

*$(\delta_1, \ldots, \delta_k)$-Round-By-Round Soundness[13]: $\Pi$ has round-by-round soundness errors $(\delta_i)_{i \in [k]}$ if there exists a state function $\mathsf{state}$ (defined below) such that for every instance $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$, round index $i \in [k]$, and a malicious prover $\tilde{\mathcal{P}}$ the following holds:*

$$\Pr\left[ \mathsf{state}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathsf{tr} \parallel \rho_i) = 1 \;:\; \begin{array}{l} \mathsf{tr} \leftarrow \tilde{\mathcal{P}} \\ \rho_i \leftarrow \{0,1\}^{r_i} \\ \mathsf{state}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0) \end{array} \right] \leq \delta_i,$$

*where $\mathsf{tr} = (\pi_1, \rho_1, \ldots, \pi_{i-1}, \rho_{i-1}, \pi_i)$ As observed in [7], if a $k$-round protocol $\Pi$ has round-by-round soundness error $\delta$, then by a union bound over the error in all the rounds, $\Pi$ has standard soundness error $k \cdot \delta$.*

*A state function is a deterministic (possibly inefficient) function $\mathsf{state}$ that receives as input an index-instance pair $(\mathbb{i}, \mathbb{x})$ and an interaction transcript $\mathsf{tr}$ and outputs a bit for which the following holds:*

1. *Empty transcript: if $\mathsf{tr} = \emptyset$, then $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0$.*
2. *Prover moves: if $\mathsf{tr} = (\pi_1, \rho_1, \ldots, \pi_i, \rho_i)$ such that $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0$, then for any $\pi_{i+1}$, $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr} || \pi_i) = 0$.*
3. *Full transcript: if $\mathsf{tr} = (\pi_1, \rho_1, \ldots, \pi_k, \rho_k)$ is a full transcript and $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0$, then the verifier outputs reject.*

The above definition is for a ternary relation $\mathcal{R} = (\mathbb{i}, \mathbb{x}, \mathbb{w})$. We get a definition for a binary relation $\mathcal{R} = (\mathbb{x}, \mathbb{w})$ by setting $\mathbb{i}$ to the empty string. For such relations, there is no need for an indexer, and the proving key $\mathsf{pp}$ and verifying key $\mathsf{vp}$ are both $\mathsf{gp}$ produced by $\mathsf{Setup}$. Computational completeness allows for the compilation of homomorphic polynomial interactive oracle proofs where the statement contains polynomial oracles (see Section 3 for details). From Theorem 5.8 in Canetti et al. [8], we can convert a public-coin multiround round-by-round sound interactive argument into a sound non-interactive argument with the Fiat-Shamir transform [18].

### 2.3 Polynomial Interactive Oracle Proofs.

Polynomial interactive oracle proofs (PIOPs) are statistically sound interactive proofs of knowledge where the prover messages are oracles to polynomials. We repeat the formal definition of PIOPs from Marlin [12] in Section A.3 with two minor generalizations. First we explicitly allow the instance to contain oracles to capture sum-check as a PIOP. Second, we allow the prover to send vectors of field elements, which is a syntactic change as the prover could just send these elements as evaluations of a polynomial. As observed in Marlin [12], without loss of generality, in every public-coin PIOP, we can postpone all verifier queries until after the interaction with the prover, thus dividing all public-coin PIOPs into an interactive phase and querying phase. PIOPs can be compiled into non-interactive arguments using a polynomial commitment scheme 2.4 and the Fiat-Shamir transform [7,12].

### 2.4 Polynomial Commitment Schemes

A polynomial commitment scheme allows a prover to commit to a polynomial and then later produce an evaluation proof for the committed polynomial at any point. We repeat the following definition from Hyperplonk [10].

**Definition 2 (Polynomial Commitment Scheme[10]).** *A polynomial commitment* $\mathsf{C}$ *for a set of polynomials* $\mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *such that:*

- $\mathsf{Setup}(1^\lambda, d) \to \mathsf{pp}$*, where* $\mathsf{pp}$ *are the public parameters to commit to a polynomial of max degree* $d$ *in each variable*
- $\mathsf{Commit}(\mathsf{pp}, f, r) \to (\mathsf{com}, r_{\mathsf{hint}})$*, where* $\mathsf{com}$ *is a public commitment to* $f$ *using some* $r$ *randomly sampled from a randomness space* $\mathcal{R}_C$ *and the (optional)* $r_{\mathsf{hint}}$ *is a secret opening hint (which might or might not be* $r$*)*
- $\mathsf{Open}(\mathsf{pp}, \mathsf{com}, x, r_{\mathsf{hint}}) \to b \in \{0, 1\}$*, where* $1$ *means that the opening of* $\mathsf{com}$ *to the message* $x$ *with opening hint* $r_{\mathsf{hint}}$ *is valid and* $0$ *means it is not*
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{com}, \boldsymbol{x}, y, d, \mu; f, r) \to b \in \{0, 1\}$ *is an interactive public-coin argument of knowledge for the (binary) relation:*

$$\mathcal{R}_{\mathsf{C}} = \left\{ (\mathsf{pp}, \mathsf{com}, \boldsymbol{x}, y); (f, r) \;:\; \begin{array}{l} f \in \mathbb{F}_\mu^{\preceq d} \wedge\ f(\boldsymbol{x}) = y \wedge\ r \in \mathcal{R}_C \\ \wedge\ \mathsf{com} = \mathsf{Commit}(\mathsf{pp}, f, r) \end{array} \right\}$$

*Polynomial commitment schemes are correct and binding and can be hiding, additively homomorphic, zero-knowledge, and extractable. We defer exact definitions to Section A.5*

KZG [23] and Zeromorph [24] are additively homomorphic polynomial commitment schemes for univariate polynomials and multivariate polynomials respectively (see Section A.5 for details). We assume they are extractable. Chiesa et al. [12] noted that we can reduce the cost of opening multiple additively homomorphic commitments at the same point to the cost of opening a single commitment.

### 2.5 Sum-Check

The sum-check protocol [25] allows a prover to prove that the sum of the evaluations of a $\mu$-variate polynomial $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$ over the boolean hypercube $\{0,1\}^\mu$ is $H$. See Figure 4 in Section A.7 for details. More precisely, sum-check is a PIOP with perfect completeness and soundness error $\mu d/\mathbb{F}$ for the following relation:

$$\mathcal{R}_{\mathsf{SUM}} = \left\{ (H, \mathcal{O}^f; f) : f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d} \ \wedge \sum_{\boldsymbol{b} \in \{0,1\}^\mu} f(\boldsymbol{b}) = H \right\}$$

If $f$ is a product of multilinear polynomials, then the prover can perform all its work in $O(2^\mu)$ time [34] [36]. To prove to a verifier that a $\mu$-variate polynomial $f$ is 0 everywhere on the hypercube [30], the prover can solicit a random vector $\boldsymbol{t} \in \mathbb{F}^\mu$ from the verifier, and then the prover and verifier run a sum-check on $f(\boldsymbol{X}) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{X})$ to convince the verifier that $\sum_{\boldsymbol{b} \in \{0,1\}^\mu} f(\boldsymbol{b}) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{b}) = 0$. This argument is perfectly complete and has knowledge error $(2d+1)\mu/|\mathbb{F}|$.

The classic sum-check PIOP is not zero-knowledge because both the evaluations sent in each round and the verifier's oracle query reveal information about $f$. To mask the evaluations sent in each sum-check round, past work [11,36] introduced the use of a random masking polynomial $g$. At the beginning of the protocol, the prover sends an oracle to $g$ and $G = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} g(\boldsymbol{b})$ to the verifier. The prover and verifier then run the sum-check protocol for a new polynomial $h = f + \alpha g$ for $\alpha \leftarrow\!\!\$\ \mathbb{F}$ sent by the verifier. Let $h_i$ be the univariate polynomial sent by the prover in round $i$, and let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_\mu)$ be the vector of verifier challenges from the first $\mu$ rounds. In the last step, the verifier queries for the evaluations of $g$ and $f$ at $\boldsymbol{\rho}$ so that it can check that $h_\mu(\rho_\mu) - \alpha g(\boldsymbol{\rho}) = f(\boldsymbol{\rho})$. The masking polynomial $g$ described Chiesa et al. [11] is potentially exponential in $\mu$, but Xie et al. [36] propose a masking polynomial of size $O(\mu d)$. They set $g(\boldsymbol{X}) = g_1(X_1) + \ldots + g_\mu(X_\mu)$ such that $g_i(X_i)$ is a random univariate polynomial of degree $d$. In practice, rather than directly commit to $g$, a prover commits to each individual $g_i$ using a ZK univariate polynomial commitment scheme (e.g., hiding KZG).

Unfortunately, the masking polynomial approach does not mask the final oracle query made by the verifier. Thus, the sum-check protocols described by Chiesa et al. [11] and Xie et al. [36] both leak one evaluation of $f$ to the verifier. In our zero-knowledge range proof construction, we cannot reveal even a single evaluation of $f$, so these prior approaches are not sufficient for our use case.

## 3 Homomorphic Polynomial Interactive Oracle Proofs

In this section we describe a new information-theoretic proof protocol called homomorphic polynomial interactive oracle proofs (HPIOPs). HPIOPs naturally extend PIOPs by enabling the verifier to make homomorphic queries over polynomial oracles. This extension explicitly mimics the functionality of instantiating a PIOP with homomorphic commitments.

Just as in PIOPs (Definition 7), prover messages in an HPIOP are restricted to oracles to polynomials and vectors of field elements. The main difference between PIOPs and HPIOPs is the types of queries that the verifier can make. In PIOPs, the verifier can query any polynomial oracles it receives at any point in the field. In HPIOPs, the verifier can make two types of queries. First, the verifier can query a linear combination of oracle polynomials at any point. Note that this allows the verifier to query a single polynomial oracle at any point just as in PIOPs. Second, the verifier can construct a polynomial and then query a polynomial oracle to check if the underlying polynomial of this oracle is a scalar multiple of the constructed polynomial. We use this property to blind sum-check polynomials. To enable the first type of query, our model allows the verifier to query groups of oracles. The syntax used in our definition borrows heavily from algebraic holographic proofs in Marlin [12].

**Definition 3 (Public-Coin Homomorphic Polynomial Interactive Oracle Proof).** *A public-coin homomorphic polynomial interactive oracle proof (HPIOP) over a field family $\mathcal{F}$ is an interactive proof for an indexed relation $\mathcal{R} = (\mathtt{i}, \mathtt{x}, \mathtt{w})$. An HPIOP is specified by a tuple of algorithms $(\mathcal{I}, \mathcal{P}, \mathcal{V})$; polynomial-time*

*computable functions* $k, s, d, \ell : \{0,1\}^* \to \mathbb{N}$*; and scalars* $\mu, n \in \mathbb{N}$*.* $\mathcal{I}, \mathcal{P}, \mathcal{V}$ *are known as indexer, prover, and verifier, respectively.* $k$ *specifies the number of rounds,* $s$ *specifies how many* $\mu$*-variate polynomial oracles are sent by the prover in each round,* $d$ *specifies the degree bounds of each polynomial oracle sent by the prover,* $\ell$ *specifies the length of the vector of field elements sent by the prover in each round,* $\mu$ *specifies the maximum number of variables for each polynomial oracle, and* $n$ *specifies the number of oracles in the instance. Both* $\mathbb{i}$ *and* $\mathbb{x}$ *can contain polynomial oracles.* $s(0)$ *is the sum of the number of polynomial oracles in both* $\mathbb{i}$ *and* $\mathbb{x}$*.*

*In the offline phase, which we call round 0, the indexer* $\mathcal{I}$ *receives as input a field* $\mathbb{F} \in \mathcal{F}$ *and an index* $\mathbb{i}$ *for* $\mathcal{R}$ *and outputs* $s(0) - n$ $\mu$*-variate polynomials* $f_{0,1}, \ldots, f_{0,s(0)-n}$*, where* $f_{0,i} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathbb{i}|,0,i)}$*.*

*In the online phase, there is an instance* $\mathbb{x}$ *and witness* $\mathbb{w}$ *such that* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$*. Let* $\mathcal{O}^{s(0)-n+1}, \ldots, \mathcal{O}^{s(0)}$ *be the* $n$ *oracles that are part of the instance* $\mathbb{x}$*, and let* $f_{0,s(0)-n+1}, \ldots, f_{0,s(0)}$ *be the corresponding polynomials, where* $f_{0,i} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathbb{i}|,0,i)}$*). The prover* $\mathcal{P}$ *receives* $(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w}, \{f_{0,i}\}_{i=1}^{s(0)})$*, and the verifier* $\mathcal{V}$ *receives* $(\mathbb{F}, \mathbb{x})$ *and oracles* $\mathcal{O}^{f_{0,i}}, \ldots, \mathcal{O}^{f_{0,s(0)}}$*.*

*The prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$ *interact in* $k = k(|\mathbb{i}|)$ *rounds. In round* $i \in [k]$*,* $\mathcal{V}$ *sends a message* $\rho_i \in \mathbb{F}^*$ *to* $\mathcal{P}$*.* $\mathcal{P}$ *sends a vector* $\boldsymbol{v}_i \in \mathbb{F}^{\ell(|\mathbb{i}|,i)}$ *and* $s(i)$ $\mu$*-variate polynomial oracles* $\mathcal{O}^{f_{i,1}}, \ldots, \mathcal{O}^{f_{i,s(i)}}$ *to* $\mathcal{V}$ *where* $f_{i,j} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|i|,i,j)}$*.*

*We say that* $(i, j)$ *is an oracle index pair if* $i \in [k]$ *and* $j \in [s(i)]$*. The verifier can make two kinds of queries any number of times at any point in the protocol. The two types of queries are as follows:*

- *Type 1: The query is a set of* $m$ *oracle index pairs* $S$*, a vector of field elements* $\boldsymbol{x} \in \mathbb{F}^\mu$*, and a vector of field elements* $\boldsymbol{c} \in \mathbb{F}^m$*. The answer is* $\sum_{i=1}^m c_i \cdot f_{S[i]}(\boldsymbol{x})$*.*
- *Type 2: The query is an oracle index pair* $(i, j)$ *and a polynomial* $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathbb{i}|,i,j)}$ *The answer is yes if* $f_{i,j}$ *is a scalar multiple* $f$*. Otherwise, the answer is abort, and the* $\mathcal{V}$ *outputs reject.*

*At the end of the interaction,* $\mathcal{V}$ *outputs accept or reject.*

*Let the verifier's output be denoted by the random variable* $\langle \mathcal{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathbb{F}, \mathbb{i}, \mathbb{x}) \rangle$*. We say an HPIOP is a public-coin HPIOP for a relation* $\mathcal{R}$ *with perfect completeness, and soundness error* $\delta$ *if the following holds:*

- *Perfect Completeness: For every field* $\mathbb{F} \in \mathcal{F}$ *and index-instance-witness tuple* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$*,*

$$\Pr\left[\langle \mathcal{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})), \mathcal{V}(\mathbb{F}, \mathbb{i}, \mathbb{x}) \rangle = 1\right] = 1$$

- *$\delta$-Soundness: For every field* $\mathbb{F} \in \mathcal{F}$*, index-instance pair* $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ *and unbounded malicious prover* $\tilde{\mathcal{P}}$*,*

$$\Pr\left[\langle \tilde{\mathcal{P}}, \mathcal{V}(\mathbb{F}, \mathbb{i}, \mathbb{x}) \rangle = 1\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)$$

- *Public-coin: All verifier messages are a uniformly random string of a prescribed length, so the verifier's randomness is its messages* $\rho_1, \ldots, \rho_k \in \mathbb{F}^*$ *and possibly additional randomness* $\rho_{k+1} \in \mathbb{F}^*$*.*

*An HPIOP may satisfy the following additional properties:*

- *Non-adaptive queries: The field element vectors and oracle index pairs that specify each verifier query are solely determined by the verifier's randomness and inputs (i.e.,* $\mathbb{F}, \mathbb{x}$*).*
- *$(\delta_1, \ldots, \delta_k)$-Round-By-Round Soundness: An HPIOP* $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *for a relation* $\mathcal{R} = (\mathbb{i}, \mathbb{x}, \mathbb{w})$ *has round-by-round soundness errors* $(\delta_i)_{i \in [k]}$ *if the HPIOP has a state function* $\mathsf{state}$ *such that for every instance* $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$*, round index* $i \in [k]$*, and malicious prover* $\tilde{\mathcal{P}}$ *the following holds:*

$$\Pr\left[\mathsf{state}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathsf{tr} \parallel \rho_i) = 1 \;:\; \begin{array}{l} \mathsf{tr} \leftarrow \tilde{\mathcal{P}} \\ \rho_i \leftarrow \{0,1\}^{r_i} \\ \mathsf{state}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathsf{tr}) = 0) \end{array}\right] \leq \delta_i,$$

*where* $\mathsf{tr} = (\mathcal{O}^{f_{0,1}}, \ldots, \mathcal{O}^{f_{0,s(0)}}, \mathcal{O}^{f_{1,1}}, \ldots, \mathcal{O}^{f_{1,s(1)}}, \boldsymbol{v}_1, \rho_1, \ldots, \mathcal{O}^{f_{i-1,1}}, \ldots, \mathcal{O}^{f_{i-1,s(i-1)}}, \boldsymbol{v}_{i-1}, \rho_{i-1}, \mathcal{O}^{f_{i,1}}, \ldots, \mathcal{O}^{f_{i,s(i)}}, \boldsymbol{v}_i)$*.*

– *Honest-Verifier Zero-Knowledge* A *HPIOP* $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *has perfect zero-knowledge if there is a PPT simulator* $\mathcal{S}$ *such that for every field* $\mathbb{F} \in \mathcal{F}$*, index-instance-witness tuple* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*, and the honest verifier* $\mathcal{V}$*, the following transcripts are identically distributed:*

$$\mathsf{View}\langle \mathcal{P}(\mathbb{F}, \mathtt{i}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathbb{F}, \mathtt{i}, \mathtt{x}) \rangle \approx \mathcal{S}^{\mathcal{V}}(\mathbb{F}, \mathtt{i}; \mathtt{x})$$

*Here the view consists of* $\mathcal{V}$*'s randomness and the list of answers to* $\mathcal{V}$*'s oracle queries.*

In this paper we only consider public-coin HPIOPs with non-adaptive queries. Just as we can divide public-coin PIOPs with non-adaptive queries into two phases [12], we can divide public-coin HPIOPs with non-adaptive queries into two phases: an interactive phase where the prover and verifier exchange messages and a query phase where the verifier makes oracle queries. In [12], they further observe that we can define the verifier by a query algorithm $\mathbb{Q}_{\mathcal{V}}$ and a decision algorithm $\mathbb{D}_{\mathcal{V}}$. Given as input the field $\mathbb{F}$, the instance $\mathtt{x}$, and randomness $\rho_1, \ldots, \rho_{k+1}$, the query algorithm outputs a query set $Q$. We express $Q$ as a tuple of two sets $(Q_1, Q_2)$ where $Q_i$ contains all the queries of Type $i$. More precisely, $Q_1$ consists of tuples of the form $(S, \boldsymbol{x}, \boldsymbol{c})$ where $S$ is a set of oracle index pairs, $\boldsymbol{x} \in \mathbb{F}^{\mu}$, and $\boldsymbol{c} \in \mathbb{F}^{|S|}$. These correspond to Type 1 verifier queries where the response is $\sum_{i=1}^{|S|} c_i \cdot f_{S[i]}(\boldsymbol{x})$. $Q_2$ consists of tuples of the form $((i,j), f)$ where $(i,j)$ is an oracle index pair and $f \in \mathbb{F}[X_1, \ldots, X_{\mu}]^{\preceq d(|\mathtt{i}|, i, j)}$. These correspond to Type 2 verifier queries where the response is yes if $f_{i,j} = \beta \cdot f$ where $\beta \in \mathbb{F}$ and abort otherwise. Given as input the field $\mathbb{F}$, the instance $\mathtt{x}$, the answers to the query set $Q$, the randomness $\rho_1, \ldots, \rho_{k+1}$, $\mathbb{D}_{\mathcal{V}}$ outputs accept or reject.

Next, we present a compilation for any HPIOP into a public-coin interactive argument of knowledge using an extractable polynomial commitment scheme $\mathsf{C}$ and a proof of knowledge $\Pi_{\mathsf{scalar}}$. We prove that if the HPIOP is round-by-round sound, $\mathsf{C}$ is extractable, and $\Pi_{\mathsf{scalar}}$ is knowledge-sound, than the compiled argument is round-by-round knowledge sound. Additionally, if $\mathsf{C}$ is hiding, $\mathsf{C.Eval}$ and $\Pi_{\mathsf{scalar}}$ are both HVZK, then the compilation preserves honest verifier zero-knowledge. Furthermore, the compiled argument can be turned into a non-interactive one via the Fiat-Shamir transformation. We emphasize the importance of proving round-by-round knowledge soundness in order to securely apply the Fiat-Shamir transformation. This is because if the cheating prover rewinds to a previous round and queries the hash function $q$ times, this round's soundness error can only increase by a factor of $q$.

**Definition 4 (HPIOP Compiler).** *Consider the following components:*

– *A* $k$*-round public coin honest-verifier zero-knowledge HPIOP specified by* $(\mathcal{I}_O, \mathcal{P}_O, \mathcal{V}_O), k, s, d, \ell, \mu$*, and* $n$ *with perfect completeness,* $(\delta_1, \ldots, \delta_k)$*-round-by-round soundness, and non-adaptive queries*
– *An extractable additively homomorphic zero-knowledge multivariate polynomial commitment scheme* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *where* $\mathsf{Eval}$ *is a zero-knowledge non-interactive argument over* $\mathbb{F}$
– *A public coin non-interactive argument of knowledge* $\Pi_{\mathsf{scalar}} = (\mathsf{Setup}_{\mathsf{scalar}}, \mathcal{I}_{\mathsf{scalar}}, \mathcal{P}_{\mathsf{scalar}}, \mathcal{V}_{\mathsf{scalar}})$ *for the following relation:*

$$\mathcal{R}^{\mathsf{C}}_{\mathsf{scalar}} = \{(\mathsf{pp}_{\mathsf{C}}; \mathsf{com}, f; \beta, r) : \mathsf{com} = \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, \beta \cdot f, r)\}$$

*Then we define* $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V})$ *as follows:*

– $\mathsf{Setup}(1^{\lambda}) \to (\mathsf{pp}_{\mathsf{C}}, \mathsf{gp}_{\mathsf{scalar}})$
  1. $D := \max\{d(N, i, j) \mid i \in \{0, 1, \ldots, k(N)\}, j \in \{1, \ldots, s(i)\}\}$
  2. $\mathsf{pp}_{\mathsf{C}} \leftarrow \mathsf{C.Setup}(1^{\lambda}, D)$
  3. $\mathsf{gp}_{\mathsf{scalar}} \leftarrow \Pi_{\mathsf{scalar}}.\mathsf{Setup}(1^{\lambda})$
  4. *Output* $(\mathsf{pp}_{\mathsf{C}}, \mathsf{gp}_{\mathsf{scalar}})$
– $\mathcal{I}(\mathsf{gp}_{\mathsf{scalar}}, \mathtt{i})$
  1. *Parse* $\mathtt{i}_{\mathsf{scalar}}, \mathtt{i}_O \leftarrow \mathtt{i}$
  2. $f_{0,1}, \ldots, f_{0,s(0)-n} \leftarrow \mathcal{I}_O(\mathbb{F}, \mathtt{i})$
  3. *For all* $j \in [s(0) - n]$*, compute* $\mathsf{com}_{0,j} \leftarrow \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, f_{0,j}, \perp)$.

4. $(\mathsf{pp}_{\mathsf{scalar}}, \mathsf{vp}_{\mathsf{scalar}}) \leftarrow \Pi_{\mathsf{scalar}}.\mathcal{I}(\mathsf{gp}_{\mathsf{scalar}}, \mathbb{i}_{\mathsf{scalar}})$
5. $\mathsf{pp} \leftarrow (\mathbb{i}, \mathsf{pp}_{\mathsf{scalar}}, \{f_{0,j}\}_{j=1}^{s(0)-n}, \{\mathsf{com}_{0,j}\}_{j=1}^{s(0)-n})$
6. $\mathsf{vp} \leftarrow (\mathsf{vp}_{\mathsf{scalar}}, \{\mathsf{com}_{0,j}\}_{j=1}^{s(0)-n})$
7. *Output* $(\mathsf{pp}, \mathsf{vp})$

– $\mathcal{P}_1(\mathsf{pp}_{\mathsf{C}}, \mathsf{gp}_{\mathsf{scalar}}, \mathbb{x}, \mathbb{w}) \to (\mathbb{x}, \mathbb{w})$

1. *Let* $f_{0,s(0)-n+1}, \ldots, f_{0,s(0)}$ *be the underlying polynomials to the* $n$ *oracle handles in the instance* $\mathbb{x}$. *For* $i \in [n]$, *compute* $\mathsf{com}_{0,s(0)-n+i} = \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, f_{0,s(0)-n+i}, r_{0,s(0)-n+i})$ *for* $r_{0,s(0)-n+i} \leftarrow\!\!\$\ \mathbb{F}$.
2. $\mathbb{x} \coloneqq (\mathbb{x}, \{\mathsf{com}_{0,j}\}_{j=s(0)-n+1}^{s(0)})$
3. $\mathbb{w} \coloneqq (\mathbb{w}, \{r_{0,j}\}_{j=s(0)-n+1}^{s(0)})$
4. *Output* $(\mathbb{x}, \mathbb{w})$

– $\langle \mathcal{P}_2(\mathsf{pp}_C, \mathsf{pp}, \mathbb{x}, \mathbb{w}, \{f_{0,i}\}_{i=1}^{s(0)}), \mathcal{V}(\mathsf{pp}_{\mathsf{C}}, \mathsf{vp}, \mathbb{x}) \rangle$*:*

- *Interactive Phase: in every round* $i \in [k]$,
  1. $\mathcal{V}$ *receives random challenge* $\rho_i$ *from* $\mathcal{V}_O$ *and forwards it to* $\mathcal{P}$
  2. $\mathcal{P}$ *forwards* $\rho_i$ *to* $\mathcal{P}_O$, *who responds with a vector* $\boldsymbol{v}_i \in \mathbb{F}^{\ell(i)}$ *and* $s(i)$ *polynomials* $f_{i,1}, \ldots, f_{i,s(i)}$ *where* $f_{i,j} \in \mathbb{F}[X_1, \ldots, X_\mu]^{d(|\mathbb{i}|,i,j)}$ *for all* $j \in [s(i)]$.
  3. $\mathcal{P}$ *sends* $\boldsymbol{v}_i$ *to* $\mathcal{V}$ *and then for each* $j \in [s(i)]$, $\mathcal{P}$ *sends a commitment* $\mathsf{com}_{i,j} = \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, p_{i,j}, r_{i,j})$, *where* $r_{i,j} \leftarrow\!\!\$\ \mathbb{F}$.
- *Query Phase:*
  1. $\mathcal{V}$ *samples* $\rho_{k+1} \leftarrow\!\!\$\ \mathbb{F}^*$ *and forwards it to* $\mathcal{P}$. $\rho_{k+1}$ *is the verifier randomness for the query phase.*
  2. $\mathcal{P}$ *computes the query set* $Q = Q_{\mathcal{V}_O}(\mathbb{F}, \mathbb{x}, \rho_1, \ldots, \rho_{k+1})$ *and parses* $(Q_1, Q_2) \leftarrow Q$.
  3. *If there is any* $((i,j), f) \in Q_2$ *such that* $f_{i,j}$ *is not a multiple of* $f$, *then* $\mathcal{P}$ *sends abort and* $\mathcal{V}$ *outputs reject.*
  4. *For all* $((i,j), f) \in Q_2$, $\mathcal{P}$ *and* $\mathcal{V}$ *initiate an execution of* $\Pi_{\mathsf{scalar}}$ *to prove that* $(\mathsf{pp}_{\mathsf{C}}, \mathsf{com}_{i,j}, f) \in \mathcal{L}(\mathcal{R}_{\mathsf{scalar}})$.
  5. $\mathcal{P}$ *initializes empty array* $A_1 = [\,]$. *For all* $(S, \boldsymbol{x}, \boldsymbol{c}) \in Q_1$, $\mathcal{P}$ *appends* $\sum_{i=1}^{|S|} c_i \cdot p_{S[i]}(\boldsymbol{x})$ *to* $A_1$. $\mathcal{P}$ *sends* $A_1$ *to* $\mathcal{V}$.
  6. *For all* $(S, \boldsymbol{x}, \boldsymbol{c}) \in Q_1$, *let* $a$ *be the corresponding answer in* $A_1$. $\mathcal{P}$ *and* $\mathcal{V}$ *initiate an execution of the* $\mathsf{C.Eval}$ *non-interactive argument to prove that* $(\mathsf{pp}_{\mathsf{C}}, \sum_{i=1}^{|S|} c_i \cdot \mathsf{com}_{S[i]}, \boldsymbol{x}, a) \in \mathcal{L}(\mathcal{R}_{\mathsf{C}})$
  7. $\mathcal{V}$ *outputs accept if and only if the results of all the non-interactive protocols in the previous two steps are accept and if* $\mathbb{D}_{\mathcal{V}_O}(\mathbb{F}, \mathbb{x}, A_1, \rho_1, \ldots, \rho_{k+1})$ *also outputs accept.*

**Theorem 1 (HPIOP Compilation Theorem).** *Let* $\mathbb{F}$ *be a field and* $\mathcal{R}$ *be an indexed relation. If* $(\mathcal{I}_O, \mathcal{P}_O, \mathcal{V}_O)$ *is a* $k$*-round public coin honest-verifier zero-knowledge HPIOP (specified by* $k, s, d, \ell, \mu$, *and* $n$*) has* $(\delta_1, \ldots, \delta_k)$*-round-by-round soundness error, and non-adaptive queries;* $\mathsf{C}$ *is an extractable additively homomorphic zero-knowledge multivariate polynomial commitment scheme, where* $\mathsf{Eval}$ *is a zero-knowledge non-interactive argument over* $\mathbb{F}$*; and* $\Pi_{\mathsf{scalar}}$ *is a public coin non-interactive argument of knowledge, then* $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V})$ *is a zero-knowledge interactive argument of knowledge with computational completeness and* $(\delta_1, \ldots, \delta_k)$*-round-by-round knowledge soundness for* $\mathcal{R}$. *Let* $\lambda \in \mathbb{N}$ *be our security parameter, and let* $N \in \mathbb{N}$ *be a size bound.*

*Proof.* We defer the round-by-round knowledge soundness proof to Section C.1 and the zero-knowledge proof to Section C.2.

## 4 Zero-Knowledge Sum-Check

Figure 1 shows a zero-knowledge sum-check protocol for a polynomial $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$. Our protocol is public-coin and has non-adaptive queries, so we describe it as an HPIOP with an interactive phase and a query phase. We describe our protocol modularly and use sub-protocols from the non-zero-knowledge sum-check described in Figure 4 for clarity.

We make three main modifications to the non-zero-knowledge sum-check in Figure 4 to achieve zero-knowledge. First, as in prior work [11,36], we use a masking polynomial $g$ to blind the evaluations revealed in each round of the sum-check. We use the more efficient blinding polynomial introduced by Xie et al. [36]. Our prover sends an oracle to $g$ along with $G = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} g(\boldsymbol{b})$ to the verifier before executing the sum-check.

Second, as noted earlier, the masking technique from prior work does not blind the single oracle query made to $f$ in the final step of sum-check, so the next modification we make is to sum over a blinded $\hat{f}$ rather than $f$ itself. The prover defines $\hat{f} := f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}$ for a random $\beta \leftarrow\$ \mathbb{F}$. Because of this single random point, a single evaluation of $\hat{f}$ is indistinguishable from random, so if the verifier receives an evaluation of $\hat{f}$ in the last step of the sum-check, the verifier learns nothing about $f$. We implement this blinding using the two types of queries permitted in HPIOPs. At the beginning of the protocol, the prover sends an oracle to the blinding polynomial $\beta \cdot \mathsf{eq}_{(1,\ldots,1)}$ to the verifier. The verifier can use a Type 2 query to check that this blinding polynomial is of the correct form. In the last step of the query phase, the verifier uses a Type 1 query to query the sum of the oracles to $f$ and $\beta \cdot \mathsf{eq}_{(1,\ldots,1)}$ to get an evaluation of $\hat{f}$.

The third modification we make to the standard sum-check protocol is to accommodate this blinding. In the sum-check from Xie et al. [36], if a prover wants to prove that the evaluations of $f$ over the hypercube sum to $H$, it solicits a random $\alpha$ from the verifier and then runs the sum-check protocol to prove that the sum of the evaluations of $f + \alpha g$ over the hypercube is $H + \alpha G$. If our prover and verifier ran a sum-check over $\hat{f} + \alpha g$, the sum would be $H + \alpha G + \beta$, which leaks $\beta$, and $\beta$ must remain unknown to the verifier. Instead, our sum-check prover claims that the sum of the evaluations of $f$ over all points on the boolean hypercube except for $(1, \ldots, 1)$ is $H$. Then, the prover and verifier execute the sum-check protocol to prove that the sum of the evaluations of $\hat{f} \cdot \mathsf{vanish}_{(1,\ldots,1)} + \alpha g$ over all points on the hypercube is $H + \alpha G$. We adjust the classic sum-check relation below in $\mathcal{R}_{\mathsf{ZKSUM}}$ to reflect this change.

**Theorem 2.** *For any degree $d \in \mathbb{N}$, Figure 1 is a public-coin honest-verifier zero-knowledge HPIOP with perfect completeness and $(\frac{d}{|\mathbb{F}|}, \ldots, \frac{d}{|\mathbb{F}|})$-round-by-round soundness for the following relation:*

$$\mathcal{R}_{\mathsf{ZKSUM}} = \left\{ (H, \mathcal{O}^f; f) : f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d} \ \wedge \sum_{\boldsymbol{b} \in \{0,1\}^\mu \setminus (1,\ldots,1)} f(\boldsymbol{b}) = H \right\}$$

*Proof.* Completeness and public-coin properties are obvious. Round-by-round soundness follows naturally from the round-by-round soundness of sum-check. We prove honest-verifier zero-knowledge below.

Honest-Verifier Zero-Knowledge Below we describe a simulator $\mathcal{S}_{\mathsf{ZKSUM}}$ with oracle access to the honest verifier $\mathcal{V}$ for any field $\mathbb{F} \in \mathcal{F}$ and instance $\mathbb{x} = (H, \mathcal{O}^f)$. The simulator answers all verifier oracle queries.
$\mathcal{S}^{\mathcal{V}}_{\mathsf{ZKSUM}}(\mathbb{F}, H, \mathcal{O}^f)$:

- Interactive Phase
  1. Send an oracle $\mathcal{O}^{\beta^* \cdot \mathsf{eq}(1,\ldots,1)}$ to $\mathcal{V}$.
  2. Sample random polynomial $g^*$ such that $g^*(\boldsymbol{X}) = g^*_1(X_1) + \ldots + g^*_\mu(X_\mu)$ where $g^*_i(X_i) = a_{i,0} + a_{i,1}X_i + \ldots a_{i,c}X_i^{d+1}$, and send oracle $\mathcal{O}^{g^*}$ and $G^* = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} g^*(\boldsymbol{b})$ to $\mathcal{V}$.
  3. Receive $\alpha \leftarrow\$ \mathbb{F} \setminus \{0, 1\}$ from $\mathcal{V}$
  4. Sample a random polynomial $f^* \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$ uniformly at random conditioned on:

$$\sum_{\boldsymbol{b} \in \{0,1\}^\mu} f^*_i(\boldsymbol{b}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{b}) = H$$

  Let $h^* = f^* + \alpha g^*$.

**Prover's Input:** $H, f$
**Verifier's Input:** $H, \mathcal{O}^f$
where $H \in \mathbb{F}$, $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$, and $\mathcal{O}^f$ is a polynomial oracle to $f$

Interactive Phase

1. $\mathsf{ZKSC.Blind}(\mathcal{P}, \mathcal{V}) \to \mathcal{P}(\beta), \mathcal{V}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}})$
   (a) $\mathcal{P}$ sends an oracle $\mathcal{O}^{\beta \cdot \mathsf{eq}(1,\ldots,1)}$ to $\mathcal{V}$ where $\beta \leftarrow\!\!\$\ \mathbb{F}$
2. $\mathsf{ZKSC.SendMask}(\mathcal{P}(d+1), \mathcal{V}) \to \mathcal{P}(g), \mathcal{V}(\mathcal{O}^g, G)$
   (a) $\mathcal{P}$ samples a random polynomial $g \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d+1}$ such that $g(\boldsymbol{X}) = g_1(X_1) + \ldots + g_\mu(X_\mu)$ where $g_i(X_i) = a_{i,0} + a_{i,1}X_i + \ldots + a_{i,d+1}X_i^{d+1}$. $\mathcal{P}$ sends the oracle $\mathcal{O}^g$ and $G = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} g(\boldsymbol{b})$ to $\mathcal{V}$.
3. $\mathsf{ZKSC.SendPolys}(\mathcal{P}(f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}, g), \mathcal{V})$
   $\to \mathcal{P}(\alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}), \mathcal{V}(\alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu})$
   (a) $\mathcal{V}$ uniformly picks $\alpha \leftarrow\!\!\$\ \mathbb{F} \setminus \{0,1\}$ and sends $\alpha$ to $\mathcal{P}$.
   (b) Let $h = (f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}) \cdot \mathsf{vanish}_{(1,\ldots,1)} + \alpha g$.
   (c) Run $\mathsf{SC.SendPolys}(\mathcal{P}(h), \mathcal{V}) \to \mathcal{P}(\{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}), \mathcal{V}(\{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu})$.

Query Phase

$\mathcal{V}\left(H, \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}, \mathcal{O}^g, G, \alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}\right)$ runs the following algorithms sequentially and outputs accept if they all output 1.

1. $\mathsf{ZKSC.BlindingCheck}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}) \to 0/1$
   (a) Query to check if $\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}$ is a scalar multiple of $\mathsf{eq}_{(1,\ldots,1)}$. If the answer is abort, output 0. Otherwise output 1.
2. $\mathsf{ZKSC.PolyCheck}(H + \alpha G, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}) \to 0/1$
   (a) Output $\mathsf{SC.PolyCheck}(H + \alpha G, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu})$.
3. $\mathsf{ZKSC.Query}(\mathcal{O}^f, \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}, \mathcal{O}^g, G, \{\rho_i\}_{i=1}^{\mu-1}, h_\mu) \to 0/1$
   (a) Sample $\rho_\mu \leftarrow\!\!\$\ \mathbb{F} \setminus \{0,1\}$. Let $\boldsymbol{\rho} := (\rho_1, \ldots, \rho_\mu)$
   (b) Query $\mathcal{O}^g$ to obtain $g(\boldsymbol{\rho})$; call this evaluation $\rho_g$.
   (c) Query the sum of $\mathcal{O}^f + \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}$ to obtain $f(\boldsymbol{\rho}) + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}(\boldsymbol{\rho})$; call this evaluation $\rho_{\hat{f}}$.
   (d) Output 1 if $\rho_{\hat{f}} \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}) + \alpha\rho_g = h_\mu(\rho_\mu)$ and 0 otherwise.

**Fig. 1.** Zero-Knowledge Sum-Check HPIOP

5. Run $\mathsf{SC.SendPolys}(\mathcal{S}(h^*), \mathcal{V}) \to \mathcal{S}(\{\rho_i^*\}_{i=1}^{\mu-1}, \{h_i^*\}_{i=1}^{\mu}), \mathcal{V}(\{\rho_i^*\}_{i=1}^{\mu-1}, \{h_i^*\}_{i=1}^{\mu})$

– Query Phase:
1. When $\mathcal{V}$ asks $\mathcal{O}$ if $\mathcal{O}^{\beta^* \cdot \mathsf{eq}_{(1,\ldots,1)}}$ is an oracle to a scalar multiple of $\mathsf{eq}_{(1,\ldots,1)}$, answer yes.
2. Let $\boldsymbol{\rho}^*$ be the vector comprised of the random challenges sent in each round of the sum-check protocol. When $\mathcal{V}$ queries $\mathcal{O}$ for the evaluation of $\mathcal{O}^{g^*}$ at $\boldsymbol{\rho}^*$, respond honestly with $g^*(\boldsymbol{\rho}^*)$.
3. When $\mathcal{V}$ queries $\mathcal{O}$ for evaluation of the sum of $\mathcal{O}^f + \mathcal{O}^{\beta^* \cdot \mathsf{eq}_{(1,\ldots,1)}}$ at $\boldsymbol{\rho}^*$, respond with $\frac{h_\mu^*(\rho_\mu^*) - \alpha g^*(\boldsymbol{\rho}^*)}{\mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}^*)}$.

To prove honest verifier zero-knowledge, we now argue that the distribution of messages sent by an honest prover when interacting with an honest verifier ("the real world") is the same as the distribution of messages sent by the simulator when interacting with an honest verifier ("the simulated world"). Because oracles are indistinguishable from one another, we only consider the distribution of the vectors sent by the prover and simulator during the interactive phase and the answers to the verifier's three queries in the query phase. Clearly, the answer to the verifier's first query is exactly the same in the real and simulated worlds. From Theorem 3 in Xie et al. [36], we have that if both the simulator and honest prover choose their masking polynomials in the way described, then all the polynomials sent during the interactive phase and the answer to the query to $\mathcal{O}^g$ in the real world are identically distributed to the polynomials sent during the interactive phase and the answer to the query to $\mathcal{O}^{g^*}$ in the simulated world. In the real world, the answer to the honest verifier's final oracle query is uniquely determined by the sum-check polynomials and the answer to the query to $\mathcal{O}^g$. Similarly, in the simulated world, the answer to the honest verifier's final oracle query is uniquely determined by the sum-check polynomials and the answer to the query to $\mathcal{O}^{g^*}$. Thus, because the sum-check polynomials and the answers to the queries to $\mathcal{O}^g$ and $\mathcal{O}^{g^*}$ are identically distributed in the real and simulated worlds, the answers to the final oracle queries are also identically distributed in the real and simulated worlds.

### 4.1 Zero-Knowledge Zero-Check

Recall the zero-check from Hyperplonk [10] discussed in Section 2.5 that proves that a polynomial $f$ is 0 at every point on the boolean hypercube. Because our sum-check effectively sums over all points in the boolean hypercube except for $(1, \ldots, 1)$, we can define a similar zero-check that proves that a polynomial $f$ is 0 at every point on the boolean hypercube except for $(1, \ldots, 1)$.

Figure 2 shows our zero-knowledge zero-check. The main difference between this protocol and the zero-knowledge sum-check is that the verifier sends a random vector $\boldsymbol{t}$ to the prover, and then the prover and verifier run the zero-knowledge sum-check protocol over $(f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}) \cdot \mathsf{eq}_{\boldsymbol{t}}$. In the query phase, the verifier performs the last check by querying $\mathcal{O}^f + \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}$ at $\boldsymbol{\rho}$ and evaluating $\mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho})$ itself. The zero-knowledge simulator is almost exactly the same, but when responding to the verifier's last query, the simulator responds with $\frac{h_\mu^*(\rho_\mu^*) - \alpha g^*(\boldsymbol{\rho}^*)}{\mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}) \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho})}$.

## 5 DekartProof: A Batch Zero-Knowledge Range Proof

We now use our zero-knowledge sum-check to describe DekartProof, a batch zero-knowledge range proof. At a high level, we wish to prove that the evaluations of a multilinear polynomial $f$ over the boolean hypercube are in $[0, n)$. If $f$ has $\log m$ variables, then we can use this technique to perform a batch range proof for $m - 1$ values.

DekartProof takes inspiration from Borgeaud's range proof for a single value [4] (Figure 5). In Borgeaud's protocol, a prover with access to a ZK, additively-homomorphic, univariate polynomial commitment scheme wants to prove that a single value $z \in [0, n)$ where $n = 2^\ell$. The protocol begins with a blinding phase and bit commitment phase:

– Blinding: Both the prover and verifier have a commitment $\mathsf{com}_f$ to $f(X) \coloneqq sX + z$ where $s \leftarrow\!\!\$\ \mathbb{F}$; the prover also knows $s$ and the randomness $r_f$ used to compute the commitment. Note that $f(0) = z$, and the coefficient $s$ of the $X$ term serves as a blinding factor to achieve zero-knowledge.

**Prover's Input:** $f$
**Verifier's Input:** $\mathcal{O}^f$
where $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$, and $\mathcal{O}^f$ is a polynomial oracle to $f$

Interactive Phase

1. $\mathsf{ZKSC.Blind}(\mathcal{P}, \mathcal{V}) \to \mathcal{P}(\beta), \mathcal{V}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}})$
2. $\mathsf{ZKSC.SendMask}(\mathcal{P}(d+2), \mathcal{V}) \to \mathcal{P}(g), \mathcal{V}(\mathcal{O}^g, G)$
3. $\mathsf{ZKZC.SendPolys}(\mathcal{P}(f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}, g), \mathcal{V})$
   $\to \mathcal{P}(\boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}), \mathcal{V}(\boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu})$
   (a) $\mathcal{V}$ uniformly samples vector $\boldsymbol{t} \leftarrow\!\!\$\ \mathbb{F}^\mu$ and sends $\boldsymbol{t}$ to $\mathcal{P}$
   (b) Run $\mathsf{ZKSC.SendPolys}(\mathcal{P}((f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}) \cdot \mathsf{eq}_{\boldsymbol{t}}, g), \mathcal{V})$
   $\to \mathcal{P}(\alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}), \mathcal{V}(\alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu})$

Query Phase

$\mathcal{V}\left(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}, \mathcal{O}^g, G, \boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\mu-1}, \{h_i\}_{i=1}^{\mu}\right)$ runs the following algorithms sequentially and outputs accept if they all output 1.

1. $\mathsf{ZKSC.BlindingCheck}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}) \to 0/1$
2. $\mathsf{ZKSC.PolyCheck}(\alpha G, \{\rho_i\}_{i=1}^{\mu-1}, \{h\}_{i=1}^{\mu}) \to 0/1$
3. $\mathsf{ZKZC.Query}(\mathcal{O}^f, \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}, \mathcal{O}^g, G, \{\rho_i\}_{i=1}^{\mu-1}, h_\mu) \to 0/1$
   (a) Sample $\rho_\mu \leftarrow\!\!\$\ \mathbb{F} \setminus \{0,1\}$. Let $\boldsymbol{\rho} \coloneqq (\rho_1, \ldots, \rho_\mu)$
   (b) Query $\mathcal{O}^g$ to obtain $g(\boldsymbol{\rho})$; call this evaluation $\rho_g$.
   (c) Query the sum of $\mathcal{O}^f + \mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}$ to obtain $f(\boldsymbol{\rho}) + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}(\boldsymbol{\rho})$; call this evaluation $\rho_{\hat{f}}$.
   (d) Output 1 if $\rho_{\hat{f}} \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}) + \alpha\rho_g = h_\mu(\rho_\mu)$ and 0 otherwise.

**Fig. 2.** Zero-Knowledge Zero-Check HPIOP

– Bit Commitment: Borgeaud's prover defines $\log n$ polynomials $f_1, \ldots, f_{\log n}$ such that each $f_i$ stores the $i^{th}$ bit of $z$ in its constant term and has a random blinding factor $s_i$ as the coefficient of its $X$ term. The prover computes commitments to each $f_i$ using randomness $r_{f_i}$

'If the prover can show that $f_i(0) \in \{0,1\}$ for all $i \in [\log n]$ and that $f(0) = \sum_{i \in [\log n]} f_i(0) \cdot 2^{i-1}$, then clearly $z \in [0, n)$. As such, the prover provides a bit proof and sum proof.

– Bit Proof: Observe that $f_i(0) \in \{0,1\}$ if and only if $f_i(X)(f_i(X)-1)$ evaluates to 0 at 0. In other words, $X$ divides $f_i(X)(f_i(X)-1)$. So to convince the verifier that $f_i(0) \in \{0,1\}$ for all $i \in [\log n]$, the prover just needs to compute and commit to $h_i(X) = \frac{f_i(X)(f_i(X)-1)}{X}$ and open all $f_i(X)$ and $h_i(X)$ at a random point $\rho$. Then the verifier can check for all $i \in [\log n]$ that $\rho \cdot h_i(\rho) = f_i(\rho)(f_i(\rho)-1)$.
– Sum Proof: The verifier can check that $f(0) = \sum_{i \in [\log n]} f_i(0) \cdot 2^{i-1}$ using the additively homomorphic properties of the commitment scheme. For this check to pass, the prover must generate both the blinding factor $s_i$'s and the commitment randomness $r_{f_i}$'s in a correlated way. More specifically, we must have that $s = \sum_{i \in [\log n]} s_i \cdot 2^{i-1}$ and that $r_f = \sum_{i \in [\log n]} r_{f_i} \cdot 2^{i-1}$.

We define a helper function $\mathsf{CorrelatedRandomness}(b, n, s)$, which outputs uniformly random $s_1, \ldots, s_n$ such that $s = \sum_{i \in [n]} s_i \cdot b^{i-1}$. The Borgeaud prover uses this function to ensure that $f(X) = \sum_{i \in [\log n]} f_i(X) \cdot 2^{i-1}$ and to ensure that the randomness used in the commitments for the $f_j$'s correctly combines to the randomness in the commitment for $f$.

For a single value $z$, Borgeaud's protocol requires $\log n$ commitments, which is not efficient. Our contribution is realizing that this protocol is efficiently batchable such that the prover only needs to produce $\log n$ commitments regardless of the number of values it wants to prove are in the range. In particular, if $f$ is a multilinear polynomial instead of a univariate polynomial, then a natural batching approach presents itself: encode each value that we want to prove is in the range at a different point on the boolean hypercube and

reserve one point on the hypercube for a random blinding value. The bit proof in our batched protocol will then prove constraints over all points in the hypercube (except the one reserved for blinding values). Using a multivariate $f$ instead of univariate $f$ allows our prover to avoid computing expensive FFTs.

Figure 3 describes $\mathsf{DekartProof}_{\mathcal{O}}$, our zero-knowledge batch range proof HPIOP. The $\mathsf{DekartProof}_{\mathcal{O}}$ prover wants to convince a verifier with oracle access to a polynomial $f$ over $\log m$ variables that $f$ evaluates to a value in $[0, n)$ at every point on the boolean hypercube except for $(1, \ldots, 1)$. Let $z_i \coloneqq f(\langle i - 1 \rangle)$ for all $i \in [m - 1]$. We adapt Borgeaud as follows, beginning with blinding and bit commitment phases.

- Blinding: The prover sends an oracle to a blinding polynomial $\beta \cdot \mathsf{eq}_{(1,\ldots,1)}$ that is used to blind $f$. The blinded polynomial is $f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)}$.
- Bit Commitment: For all $j \in [\log n]$, the prover defines $\hat{f}_j$ so that it stores the $j^{th}$ bits of each $z_i$. That is, for all points on the hypercube not equal to $(1, \ldots, 1)$, $\hat{f}_j$ is equal to the $j^{th}$ bit of $z_i$ at the $i^{th}$ point on the hypercube, and at $(1, \ldots, 1)$, $\hat{f}_j$ is equal to a random blinding value. As in Borgeaud's protocol, our prover uses the $\mathsf{CorrelatedRandomness}$ function to generate these random blinding values to ensure that they correctly combine to $\beta$. The prover sends oracles to all $\hat{f}_j$'s to the verifier.

Just like the Borgeaud prover, our prover now also needs to create a sum proof and bit proof to show that (1) for all $j \in [\log n]$ and for all points $\boldsymbol{b}$ on the boolean hypercube except for $(1, \ldots, 1)$, $\hat{f}_j(\boldsymbol{b}) \in \{0, 1\}$ and (2) $f + \beta \cdot \mathsf{eq}_{(1,\ldots,1)} = \left(\sum_{i=1}^{\log n} \hat{f}_i \cdot 2^{i-1}\right)$. These are essentially the same constraints proved by the Borgeaud prover. The main difference is that our prover needs to prove that multiple evaluations of the $\hat{f}_j$'s are in $\{0, 1\}$, whereas the Borgeaud prover only needs to prove that a single evaluation of each $f_j$ is in $\{0, 1\}$.

- Bit Proof: We define the same helper polynomial as before: $\hat{f}_j(\hat{f}_j - 1)$. Observe that $\hat{f}_j$ evaluates to $\{0, 1\}$ everywhere on the hypercube except for at $(1, \ldots, 1)$ if and only if $\hat{f}_j(\hat{f}_j - 1)$ is 0 everywhere on the hypercube except for at $(1, \ldots, 1)$. We can prove this statement about $\hat{f}_j(\hat{f}_j - 1)$ with a direct application of a zero-check. Instead of running $\log n$ separate zero-checks for each $\hat{f}_j(\hat{f}_j - 1)$, the prover and verifier run a zero-check on a random linear combination of all the $\hat{f}_j(\hat{f}_j - 1)$'s. This zero-check differs from the general one described in Section 4.1 in two ways. First, because the $\hat{f}_j$'s are already blinded, this instantiation of the zero-knowledge zero-check does not include the step where the prover sends oracles to the blinding polynomials. This is an optimization that prevents the prover from having to reblind the $\hat{f}_j$'s. Second, since the zero-check is being performed over a composition of the $\hat{f}_j$'s (e.g., rather than a single blinded $\hat{f}$), in the last step, the verifier queries the oracles to all $\hat{f}_j$'s at the point $\boldsymbol{\rho}$ rather than querying a single oracle.
- Sum Proof: Although our prover could use the homomorphism of the commitments for the sum proof like the Borgeaud prover does, we instead use a more efficient equality check. By the Schwartz-Zippel Lemma [29,40], if $f + \beta \cdot \mathsf{eq}_{(1,\cdots,1)}$ and $\sum_{i=1}^{\log n} \hat{f}_j \cdot 2^{j-1}$ are equal at a random point, then these two polynomials are equal with very high probability. Crucially, from the last round of the zero-check from the bit proof, the verifier already has evaluations $\rho_{\hat{f}_j}$'s of the $\hat{f}_j$'s at $\boldsymbol{\rho}$, so it only needs to query for the evaluation of $f + \beta \cdot \mathsf{eq}_{(1,\cdots,1)}$ at $\boldsymbol{\rho}$ and check if it is equal to $\sum_{j=1}^{\log n} \rho_{\hat{f}_j} \cdot 2^{j-1}$. As noted in Section 2.4, we can open multiple homomorphic polynomial commitments for the same cost as opening a single commitment, so when we compile our HPIOP, this additional opening is almost free.

**Theorem 3.** $\mathsf{DekartProof}_{\mathcal{O}}$ *(Figure 3) is a public-coin honest-verifier zero-knowledge HPIOP with perfect completeness and* $(\delta, \ldots, \delta_k)$*-round-by-round soundness for the following relation:*

$$\mathcal{R}_{\mathsf{ZKRANGE}} = \left\{ (n, m, \mathcal{O}_f; f) : \forall \boldsymbol{b} \in \{0,1\}^{\log m} \setminus (1, \ldots, 1), f(\boldsymbol{b}) \in [0, n) \right\}$$

*where* $\delta_1 < \frac{1}{|\mathbb{F}|}$, $\delta_2 = 0$, *and* $\forall i \in [3, k]\, \delta_i < \frac{d}{|\mathbb{F}|}$.

**Prover's Input:** $n, m, f$
**Verifier's Input:** $n, m, \mathcal{O}^f$
where $n, m \in \mathbb{N}$, $f \in \mathbb{F}[X_1, \ldots, X_{\log m}]^{\preceq 1}$, and $\mathcal{O}^f$ is a polynomial oracle to $f$
Interactive Phase:

1. Run $\mathsf{ZKSC.Blind}(\mathcal{P}, \mathcal{V}) \to \mathcal{P}(\beta), \mathcal{V}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}})$
2. For all $i \in [m-1]$, $\mathcal{P}$ sets $z_{i,1}, \ldots, z_{i,\log n} \leftarrow f(\langle i-1 \rangle)$.
3. $\mathcal{P}$ sets $\beta_1, \ldots, \beta_{\log n} \leftarrow \mathsf{CorrelatedRandomness}(2, \log n, \beta)$.
4. For all $j \in [\log n]$, $\mathcal{P}$ defines $\hat{f}_j := \beta_j \cdot \mathsf{eq}_{(1,\ldots,1)} + \sum_{i \in [m-1]} z_{i,j} \cdot \mathsf{eq}_{\langle i-1 \rangle}$
5. $\mathcal{P}$ sends oracles $\mathcal{O}^{\hat{f}_1}, \ldots, \mathcal{O}^{\hat{f}_{\log n}}$ to $\mathcal{V}$ and $\hat{f}_1, \ldots, \hat{f}_{\log n}$ to $\mathcal{O}$.
6. $\mathcal{V}$ sends $\gamma_1, \ldots, \gamma_{\log n} \leftarrow\!\!\$\ \mathbb{F}$ to $\mathcal{P}$.
7. $\mathsf{ZKZC.SendMask}(\mathcal{P}(4), \mathcal{V}) \to \mathcal{P}(g), \mathcal{V}(\mathcal{O}^g, G)$
8. $\mathsf{ZKZC.SendPolys}(\mathcal{P}\left(\left(\sum_{j=1}^{\log n} \gamma_j \hat{f}_j(\boldsymbol{b})(\hat{f}_j(\boldsymbol{b}) - 1)\right), g\right), \mathcal{V}) \to$
   $\mathcal{P}(\boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\log m - 1}, \{h_i\}_{i=1}^{\log m}), \mathcal{V}(\boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\log m - 1}, \{h_i\}_{i=1}^{\log m})$

Query Phase:
$\mathcal{V}\left(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}}, \{\mathcal{O}^{\hat{f}_i}, \gamma_i\}_{i=1}^{\log n}, \mathcal{O}^g, G, \boldsymbol{t}, \alpha, \{\rho_i\}_{i=1}^{\log m - 1}, \{h_i\}_{i=1}^{\log m}\right)$:

1. If $\mathsf{ZKSC.BlindingCheck}(\mathcal{O}^{\beta \cdot \mathsf{eq}_{(1,\ldots,1)}} = 0$, $\mathcal{V}$ outputs reject.
2. If $\mathsf{ZKSC.PolyCheck}(\alpha G, \{\rho_i\}_{i=1}^{\log m - 1}, \{h_i\}_{i=1}^{\log m}) = 0$, $\mathcal{V}$ outputs reject.
3. $\mathcal{V}$ samples $\rho_{\log m} \leftarrow\!\!\$\ \mathbb{F} \setminus \{0, 1\}$. Let $\boldsymbol{\rho} := (\rho_1, \ldots, \rho_{\log m})$. $\mathcal{V}$ queries $\mathcal{O}^g$ at $\boldsymbol{\rho}$ to get $\rho_g = g(\boldsymbol{\rho})$. Then for all $j \in [\log n]$, $\mathcal{V}$ queries $\mathcal{O}^{\hat{f}_j}$ at $\boldsymbol{\rho}$ to get $\rho_{\hat{f}_j}$. $\mathcal{V}$ checks if
$$\left(\sum_{j \in [\log n]} \gamma_j \rho_{\hat{f}_j}(\rho_{\hat{f}_j} - 1)\right) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}) + \alpha \rho_g = h_{\log m}(\rho_{\log m})$$
.
4. $\mathcal{V}$ queries $\mathcal{O}$ for the evaluation of $\mathcal{O}^f + \mathcal{O}^\beta$ at $\boldsymbol{\rho}$ and checks that this evaluation is equal to $\sum_{j=1}^{\log n} \rho_{\hat{f}_j} \cdot 2^{j-1}$. If not, $\mathcal{V}$ rejects. Else, $\mathcal{V}$ accepts.

**Fig. 3.** $\mathsf{DekartProof}_{\mathcal{O}}$: A Zero-Knowledge HPIOP to Prove $m-1$ Values in $[0, n)$

*Proof.* Completeness is obvious. We prove round-by-round soundness and ZK below.

Round-by-Round Soundness: We denote the state function as $\mathsf{state}$.

**Step 0. Empty transcript.** Given a polynomial $f$, we set $\mathsf{state}(\mathcal{O}^f, \emptyset) = 1$ if and only if $\mathcal{O}^f$ evaluates to a value in $[0, n)$ at every point in $\{0,1\}^{\log m} \setminus (1, \ldots, 1)$.

**Step 1.** The interaction starts with the prover sending $\mathcal{O}^{\beta \cdot \mathsf{eq}(1,\ldots,1)}, \mathcal{O}^{\hat{f}_1}, \ldots, \mathcal{O}^{\hat{f}_{\log n}}$ to the verifier. We set $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\gamma_1, \ldots, \gamma_{\log n}) = 1$ if and only if the following conditions hold:

1. $\forall \boldsymbol{b} \in \{0,1\}^{\log m} \setminus (1, \ldots, 1) : \sum_{j \in [\log n]} \gamma_j \hat{f}_j(\boldsymbol{b})(\hat{f}_j(\boldsymbol{b}) - 1) = 0$
2. There exists $\beta$ such that $\mathcal{O}^{\beta \cdot \mathsf{eq}(1,\ldots,1)}$ is an oracle for $\beta \cdot \mathsf{eq}(1, \ldots, 1)$

We can bound the following probability:

$$\Pr_{\gamma_1, \ldots, \gamma_{\log n}}[\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\gamma_1, \ldots, \gamma_{\log n}) = 1 | \mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0] \leq \frac{1}{|\mathbb{F}|}.$$

Suppose $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$, so there exists at least one $\hat{f}_j$ that was not computed as prescribed. Then, the sum is 0 only if vectors $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{\log_n})$ and $\hat{\boldsymbol{f}}(\hat{\boldsymbol{f}} - \mathbf{1}) = (\hat{f}_1(\hat{f}_1 - 1), \ldots, \hat{f}_{\log n}(\hat{f}_{\log n} - 1))$ are orthogonal. The probability of the verifier sending such $\boldsymbol{\gamma}$ is at most $\frac{1}{|\mathbb{F}|}$.

**Step 2.** Next, in $\mathsf{ZKZC.SendMask}$, the prover sends $\mathcal{O}^g$ and $G$, so the transcript has the following form:

$$\mathsf{tr} = (\mathcal{O}^{\beta \cdot \mathsf{eq}(1,\ldots,1)}, (\mathcal{O}^{\hat{f}_i})_{i \in [n]}, (\gamma_i)_{i \in [n]}, \mathcal{O}^g, G).$$

We set $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\boldsymbol{t}, \lambda) = 1$ if and only if the following hold:

1. $\alpha G = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} \left( \sum_{j \in [\log n]} \gamma_j \hat{f}_j(\boldsymbol{b})(\hat{f}_j(\boldsymbol{b}) - 1) \right) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{b}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{b}) + \alpha g(\boldsymbol{b})$
2. $G = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} g(\boldsymbol{b})$
3. There exists $\beta$ such that $\mathcal{O}^{\beta \cdot \mathsf{eq}(1,\ldots,1)}$ is an oracle for $\beta \cdot \mathsf{eq}(1, \ldots, 1)$

Suppose $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$. Then, according to the state function definition, the following holds:

$$A = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m} \setminus (1,\ldots,1)} \sum_{j \in [\log n]} \gamma_j \hat{f}(\boldsymbol{b})_j (\hat{f}_j \boldsymbol{b} - 1) \neq 0.$$

Using this notation, condition in Item 1 can be rewritten as:

$$\alpha G = A + \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} \alpha g(\boldsymbol{b}).$$

Subtracting condition in Item 2, we get:

$$\alpha G - \alpha G = A + \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} \alpha g(\boldsymbol{b}) - \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} \alpha g(\boldsymbol{b}) \rightarrow A = 0$$

But $A$ is not equal to 0, so we get a contradiction. Therefore,

$$\Pr_{\boldsymbol{t}, \lambda}[\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\boldsymbol{t}, \lambda) = 1 | \mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0] = 0.$$

**Step 3. First round of the sum-check.** In the first round of the sum-check, in $\mathsf{ZKZC.SendPolys}$, the prover sends a univariate polynomial $h_1$ and the verifier sends randomness $\rho_1$. We set $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_1) = 1$ if and only if

$$\alpha G = h_1(0) + h_1(1).$$

We can show that

$$\Pr_{\rho_1}\left[\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_1) = 1 | \mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0\right] \leq d/\mathbb{F}.$$

Suppose $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$. Then,

$$\alpha G \not\equiv \sum_{\boldsymbol{b}\in\{0,1\}^{\log m}} \left( \sum_{j\in[\log n]} \gamma_j \hat{f}_j(\boldsymbol{b})(\hat{f}_j(\boldsymbol{b}) - 1) \right) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{b}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{b}) + \alpha g(\boldsymbol{b}).$$

Therefore, $h_1$ was not computed as prescribed. But two univariate polynomials of degree at most $d$ can agree on $d$ points at most, therefore,

$$\delta_i \leq d/\mathbb{F}.$$

**Step 4. Sum-check round $i > 1$.** In each round of the sum-check, the prover sends a univariate polynomial $h_i$ and the verifier sends randomness $\rho_i$. We set $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_i) = 1$ if and only if

$$h_i(0) + h_i(1) = h_{i-1}(\rho_i).$$

Again, we can show that

$$\Pr_{\rho_i}\left[\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_i) = 1 | \mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0\right] = \delta_i \leq d/\mathbb{F}.$$

Suppose $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$. Therefore, similar to the previous step argument, $h_i$ was not computed correctly. However, two univariate polynomials of degree at most $d$ can agree on $d$ points at most, therefore,

$$\delta_i \leq d/\mathbb{F}.$$

**Step 5. Last round of sum-check round.** In the end of sum-check, the prover sends a univariate polynomial $h_{\log_m}$. The verifier samples $\rho_{\log_m}$, queries $\mathcal{O}^g$ and $\mathcal{O}^{\hat{f}_j}$'s at $\rho$. We set $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_{\log_m}) = 1$ if and only if

$$\left( \sum_{j\in[\log n]} \gamma_j \rho_{\hat{f}_j}(\rho_{\hat{f}_j} - 1) \right) \cdot \mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho}) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}) + \alpha\rho_g = h_{\log m}(\rho_{\log m})$$

We show

$$\Pr_{\rho_i}\left[\mathsf{state}(\mathcal{O}^f, \mathsf{tr}||\rho_{\log_m}) = 1 | \mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0\right] = \delta_i \leq d/\mathbb{F}.$$

Suppose $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$. Therefore, similar to the previous step argument,

$$\delta_i \leq d/\mathbb{F}.$$

**Step 5. Verifier's decision.** We need to show that if $\mathsf{state}(\mathcal{O}^f, \mathsf{tr}) = 0$, then the verifier rejects. Let us analyze the verifier's algorithm:

1. $\mathsf{ZKSC.BlindingCheck}$ is checked in Steps 1-2.
2. $\mathsf{ZKSC}$ is checked in Steps 3-4.
3. Verifier's check 3 is checked in Step 2.
4. In verifier's check 4, the following must hold for the verifier to accept:
   (a) Evaluation of $\mathcal{O}^f + \mathcal{O}^{\beta\cdot\mathsf{eq}(1,\ldots,1)}$ at $\boldsymbol{\rho}$ is $f(\boldsymbol{\rho})$ since $\mathcal{O}^{\beta\cdot\mathsf{eq}(1,\ldots,1)}$ is an oracle to $\beta \cdot \mathsf{eq}(1,\ldots,1)$, which is checked in Item 1.
   (b) If $f(\rho) \neq \sum_{j=1}^{\log n} \rho_{\hat{f}_j} \cdot 2^{j-1}$, then $\hat{f}_j$'s were not computed correctly. This is checked in Step 2.

Honest-Verifier Zero-Knowledge: We define our simulator $\mathcal{S}_{\mathsf{ZKRANGE}}$ with oracle access to the honest verifier $\mathcal{V}$ as follows:
$\mathcal{S}^{\mathcal{V}}_{\mathsf{ZKRANGE}}(n, m, \mathcal{O}^f)$:

– Interactive Phase
  1. Send an oracle $\mathcal{O}^{\beta^*\cdot\mathsf{eq}(1,\ldots,1)}$ to $\mathcal{V}$.

2. Select polynomials $\hat{f}_1^*, \ldots, \hat{f}_{\log n}^* \in \mathbb{F}[X_1, \ldots, X_{\log m}]^{\preceq 1}$ uniformly at random conditioned on $\hat{f}_j^*(\boldsymbol{b}) \in \{0,1\}$ for all $\boldsymbol{b} \in \{0,1\}^\mu \setminus (1, \ldots, 1)$ for all $j \in [\log n]$. Send oracles $\mathcal{O}^{\hat{f}_1^*}, \ldots, \mathcal{O}^{\hat{f}_{\log n}^*}$ to $\mathcal{V}$.
3. Receive random $\gamma_1, \ldots, \gamma_{\log n} \leftarrow\!\!\$\ \mathbb{F}$ from $\mathcal{V}$
4. Now simulate the zero-check.
   (a) Sample random polynomial $g^*$ such that $g^*(\boldsymbol{X}) = g_1^*(X_1) + \ldots + g_{\log m}^*(X_{\log m})$ where $g_i^*(X_i) = \sum_{j=0}^4 a_{i,j}^* X_i^j$, and send an oracle $\mathcal{O}^{g^*}$ and $G^* = \sum_{\boldsymbol{b} \in \{0,1\}^{\log m}} g^*(\boldsymbol{b})$ to $\mathcal{V}$.
   (b) Receive random $\alpha \leftarrow\!\!\$\ \mathbb{F} \setminus \{0,1\}$ and $\boldsymbol{t} \leftarrow\!\!\$\ \mathbb{F}^{\log m}$.
   (c) Let $h^* = \left(\sum_{j \in [\log n]} \gamma_j \hat{f}_j^*(\hat{f}_j^* - 1)\right) \cdot \mathsf{eq}_{\boldsymbol{t}} \cdot \mathsf{vanish}_{(1,\ldots,1)} + \alpha g^*$. Run $\mathsf{SC.SendPolys}(\mathcal{S}(h^*), \mathcal{V}) \rightarrow \mathcal{S}(\{\rho_i^*\}_{i=1}^{\mu-1}, \{\boldsymbol{v}_i^*\}_{i=1}^{\mu}), \mathcal{V}(\{\rho_i^*\}_{i=1}^{\mu-1}, \{\boldsymbol{v}_i^*\}_{i=1}^{\mu})$

– Query Phase
1. When $\mathcal{V}$ asks $\mathcal{O}$ if $\mathcal{O}^{\beta^* \cdot \mathsf{eq}_{(1,\ldots,1)}}$ is an oracle to a scalar multiple of $\mathsf{eq}_{(1,\ldots,1)}$, answer yes.
2. Let $\boldsymbol{\rho}^*$ be the vector comprised of the random challenges sent in each round of the sum-check protocol. When $\mathcal{V}$ queries $\mathcal{O}$ for the evaluation of $\mathcal{O}^{g^*}$ at $\boldsymbol{\rho}^*$, respond honestly with $g^*(\boldsymbol{\rho}^*)$.
3. When $\mathcal{V}$ queries $\mathcal{O}$ for the evaluations of $\mathcal{O}^{\hat{f}_1^*}, \ldots, \mathcal{O}^{\hat{f}_{\log n}^*}$ at $\boldsymbol{\rho}^*$ for $j \in [\log n]$, sample random $\rho_{\hat{f}_j^*} \leftarrow\!\!\$\ \mathbb{F}$ for $j \in [2, \log n + 1)$. Then try to solve the following equation for $\rho_{\hat{f}_1^*}$:

$$\rho_{\hat{f}_1^*}(\rho_{\hat{f}_1^*} - 1) = \frac{1}{\gamma_1}\left(\frac{h_\mu(\rho_\mu) - \alpha g^*(\boldsymbol{\rho}^*)}{\mathsf{eq}_{\boldsymbol{t}}(\boldsymbol{\rho}^*) \cdot \mathsf{vanish}_{(1,\ldots,1)}(\boldsymbol{\rho}^*)} - \sum_{j \in [2, \log n + 1)} \gamma_j \rho_{\hat{f}_j^*}(\rho_{\hat{f}_j^*} - 1)\right)$$

   If there is no solution, rerun the simulator from the beginning.
4. When $\mathcal{V}$ queries $\mathcal{O}$ for the evaluation of $\mathcal{O}^f + \mathcal{O}^{\beta^* \cdot \mathsf{eq}_{(1,\ldots,1)}}$ at $\boldsymbol{\rho}^*$, respond with $\sum_{j=1}^{\log n} \rho_{\hat{f}_j^*} \cdot 2^{j-1}$.

We now argue that there is a solution for $\rho_{\hat{f}_1^*}$ in Step 3 of the simulation query phase with probability $1/2$, so the simulator will run an expected number of 2 times. Observe that $\gamma_1$ is a random field element chosen independently of everything else on the RHS of the equation, which means that the RHS of the equation is a random element in $\mathbb{F}$. Let's call this RHS $c$. Then the simulator needs to solve an equation $\rho_{\hat{f}_1^*}^2 - \rho_{\hat{f}_1^*} - c = 0$. There is a solution for $\rho_{\hat{f}_1^*}$ if the determinant $\sqrt{1 + 4c}$ is a quadratic residue in $\mathbb{F}$. Because $\mathbb{F}$ is a prime field, $1 + 4c$ is a one-to-one mapping from $\mathbb{F} \rightarrow \mathbb{F}$, so since a random element from $\mathbb{F}$ is a quadratic residue with probability $1/2$, $1 + 4c$ is also a quadratic residue with probability $1/2$, which means that there is a solution for $\rho_{\hat{f}_1^*}$ with probability $1/2$.

To prove honest verifier zero-knowledge, we now argue that the distribution of messages sent by an honest prover when interacting with an honest verifier ("the real world") is the same as the distribution of messages sent by the simulator ("the simulated world"). This proof is very similar to the zero-knowledge proof from the previous section. Because oracles are indistinguishable from one another, we only consider the distribution of the vectors sent by the prover or simulator during the interactive phase and the answers to the verifier's queries in the query phase. Clearly, the answer to the verifier's first query is exactly the same in the real and simulated worlds. From Theorem 3 in Xie et al. [36], we have that if both the simulator and honest prover choose their masking polynomials in the way described, then all the vectors sent during the interactive phase and the answer to the query to $\mathcal{O}^g$ in the real world are identically distributed to the vectors sent during the interactive phase and the answer to the query to $\mathcal{O}^{g^*}$ in the simulated world.

We now discuss why the joint distribution of query answers in Steps 3 and 4 of the real world is indistinguishable from the joint distribution of query answers in Steps 3 and 4 in the simulated world. We first argue that the distribution of query answers in Step 3 of the real world is indistinguishable from the distribution of query answers in Step 3 of the simulated world. Then we argue that, conditioned on these two distributions being the same, the distribution of query answers of Step 4 in the real world is indistinguishable from the distribution of query answers of Step 4 in the simulated world.

Observe that in the real world, the honest prover generates the correlated random $\beta_j$'s with respect to $\beta$ such that without $\beta$, the set of $\beta_j$'s is indistinguishable from a set of $\log n$ random field elements. For

this reason, a set containing one evaluation from each $\hat{f}_j$ (blinded by $\beta_j$) is also indistinguishable from $\log n$ random field elements. Thus, the evaluations of the $\hat{f}_j$'s revealed in Step 3 in the real world are uniformly random conditioned on the equation in Step 3 being satisfied. The right hand side of the equation in Step 3 is exactly determined by the sum-check vectors sent by the prover in the interactive phase. In the simulated world, the query answers given in Step 3 are sampled to be uniformly random conditioned on the equation in Step 3 from the real world being satisfied. The right hand side of this equation is determined by the sum-check vectors sent by the simulator in the interactive phase, which we already argued are indistinguishable from the sum-check vectors sent by the honest prover. This means that the query answers sent by the honest prover in Step 3 of the real world are indistinguishable from the query answers sent by the simulator in Step 3 of the simulated world. Finally, the final query answer in Step 4 of the real world is completely determined by the query answers from Step 3 of the real world, and similarly, the final query answer in Step 4 of the simulated world is completely determined by the query answers from Step 3 of the simulated world. Because the distribution of query answers from Step 3 of the real world is indistinguishable from the distribution of query answers from Step 3 of the simulated world, the query answer in Step 4 in the real world is thus indistinguishable from the query answer in Step 4 of the simulated world.

We obtain $\mathsf{DekartProof}$ by compiling $\mathsf{DekartProof}_{\mathcal{O}}$ with a hiding, extractable, zero-knowledge commitment scheme $\mathsf{C}$ and an argument of knowledge $\Pi^{\mathsf{C}}_{\mathsf{scalar}}$ for $\mathcal{R}^{\mathsf{C}}_{\mathsf{scalar}}$. More formally:

**Theorem 4.** *If the polynomial commitment scheme* $\mathsf{C}$ *is hiding, extractable, and* $\mathsf{C.Eval}$ *is honest-verifier zero-knowledge and* $\Pi^{\mathsf{C}}_{\mathsf{scalar}}$ *is a argument of knowledge for* $\mathcal{R}^{\mathsf{C}}_{\mathsf{scalar}}$*, then* $\mathsf{DekartProof}$*, the output of compiling* $\mathsf{DekartProof}_{\mathcal{O}}$ *with* $\mathsf{C}$ *and* $\Pi^{\mathsf{C}}_{\mathsf{scalar}}$*, is an honest-verifier zero-knowledge argument of knowledge for the following relation:*

$$\left\{ n, m, \mathsf{com}_f; f, r : \begin{array}{l} \mathsf{com}_f = \mathsf{C.Commit}(f, r) \\ \wedge \forall \boldsymbol{b} \in \{0,1\}^{\log m} \setminus (1, \ldots, 1), f(\boldsymbol{b}) \in [0, n) \end{array} \right\}$$

## 6 Performance Analysis

In Table 1, we compare the costs of our zero-knowledge range proof with the costs of other state-of-the-art zero-knowledge batch range proofs. Our cost estimates come from compiling $\mathsf{DekartProof}_{\mathcal{O}}$ with Zeromorph ($\mathsf{ZM}$) for the multilinear polynomial commitments, hiding KZG for the univariate polynomial commitments for the blinding polynomial, and an argument of knowledge $\Pi^{\mathsf{ZM}}_{\mathsf{scalar}}$ for $\mathcal{R}^{\mathsf{ZM}}_{\mathsf{scalar}}$ as defined in Section B.1.

Our Costs Explained: The main prover work is committing to $\hat{f}_1, \ldots, \hat{f}_{\log n}$ with Zeromorph and doing the prover work for a sum-check over a $\log m$-variate polynomial of degree 4. Each $\hat{f}_i$ is a $\log m$-variate polynomial, so committing to $\log n$ of them with Zeromorph takes $m \log n$ $\mathbb{G}_1$ work for $\log n$ multiexponentiations of size $m$. We note that only $\log n$ of the exponents involved in these multiexponentiations are random elements in the field, and $(m-1)\log n$ of the exponents are either 0 or 1, which dramatically speeds up the computation in practice. Using techniques from Libra [36], the sum-check work can be calculated with $O(m \log n)\mathbb{F}$ work. All together, that is $O(m \log n)\mathbb{G}_1, O(m \log n)\mathbb{F}$. Instead of opening the commitments to each $\hat{f}_j$ and $g_i$ individually, we use the trick from Section 2.4 to batch open additively homomorphic commitments, so the prover only has to calculate random linear combinations of the commitments and then open a single Zeromorph commitment and single KZG commitment, which does not affect the asymptotic cost.

The main verifier work is performing the sum-check verifier work for a $\log m$-variate polynomial of degree 4, checking that the $\rho_{\hat{f}_j}$'s sum up to the correct value, and verifying the Zeromorph and KZG commitment openings. The sum-check verifier work can be done in $O(\log m)$ $\mathbb{F}$ operations, and checking the sum of the $\rho_{\hat{f}_j}$'s takes $O(\log n)$ $\mathbb{F}$ operations. To check the commitment openings, the verifier calculates a random linear combination of $\log n + 1$ Zeromorph commitments, a random linear combination of $\log m$ KZG commitments, and then verifies both proofs. This takes $O(\log m + \log n)$ $\mathbb{G}_1$ work, $O(1)$ $\mathbb{G}_2$ work, and 5 pairings. All together that is $O(\log n + \log m)$ $\mathbb{F}$, $O(\log m + \log n)$ $\mathbb{G}_1$, $O(1)$ $\mathbb{G}_2$, and 5 pairings.

The main contributors to proof size are the $\log n$ Zeromorph commitments, the $\log m$ KZG commitments, the sum-check field elements for a sum-check over a $\log m$-variate polynomial with degree 4, and the Zeromorph opening proof. Each commitment is a single $\mathbb{G}_1$ element, so in total, the commitments contribute $\log n + \log m$ $\mathbb{G}_1$ elements. The sum-check consists of $5 \log m$ $\mathbb{F}$ elements, and the Zeromorph opening proof is $\log m + 5$ $\mathbb{G}_1$ elements. The total proof size is thus $O(\log m)$ $\mathbb{F}$, $O(\log m + \log n)$ $\mathbb{G}_1$.

From Table 1, we see that Lasso instantiated with Sona (a polynomial commitment scheme from [32]) outperforms our construction. However, this Lasso instantiation is not zero-knowledge, and we believe that making Lasso zero-knowledge will increase its costs significantly. The two options for making Lasso zero-knowledge are to take a recursive approach or to directly make the Lasso protocol zero-knowledge. The former requires a prover to prove knowledge of a correct Lasso proof in zero-knowledge, which will be more expensive than a custom proof like ours. The latter approach is non-trivial, and we briefly explain why we believe it would result in a slower verifier or larger proof sizes. Lasso's prover invokes an optimized version of GKR [22] on part of a circuit with $m$ inputs, so to make the Lasso protocol zero-knowledge, we would need to use a zero-knowledge version of GKR. Libra [36] describes an efficient zero-knowledge GKR protocol, but its verifier calculates $\log m$ pairings for an input layer with $m$ gates. In contrast, our verifier uses only 5 pairings regardless of the size of $m$. Virgo [39] replaces Libra's pairing-based polynomial commitment scheme with a transparent hash-based scheme, resulting in a faster prover and verifier but much larger proof sizes.

Bulletproofs have smaller proof sizes but a slower verifier than DekartProof. Also, while the DekartProof and Bulletproofs provers have the same asymptotic costs, the Bulletproofs prover computes a multiexponentiation of size $2m \log n + 1$ where the exponents are random field elements. In our multiexponentiations, $(m-1) \log n$ of $m \log n$ exponents are 0 or 1, so our prover should be concretely faster in computing these multiexponentiations. The construction described in MissileProof is not zero-knowledge as is, but we believe it can be made zero-knowledge without affecting its asymptotic costs. MissileProof has a smaller proof size and faster verifier, but our construction has a faster prover as we work with multilinear polynomials and avoid calculating FFTs. In contrast, the MissileProof prover works over univariate polynomials and must compute FFTs.

## 7 Conclusion and Open Problems

We introduced DekartProof, a new batched zero-knowledge range proof. Our construction builds on well-established primitives, including polynomial commitment schemes and the sum-check protocol. In particular, we develop a zero-knowledge variant of the sum-check protocol and propose a new information-theoretic primitive, a homomorphic PIOP, for which we also provide a compiler to SNARKs. Our techniques offer an efficient alternative to existing range proofs and lookup-based methods, with applications in publicly verifiable secret sharing, confidential transactions, and secure election protocols.

An interesting direction for future work is to design a construction that preserves our prover's efficiency while reducing the verifier's work, and achieving a short proof size.

**Acknowledgments.** The authors thank Justin Thaler and Patrick Towa for fruitful discussions. This work was funded by DARPA, the Simons Foundation, and UBRI. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

## References

1. T. Attema, R. Cramer, and S. Fehr. Compressing proofs of k-out-of-n partial knowledge. In *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV*, page 65–91, Berlin, Heidelberg, 2021. Springer-Verlag.
2. D. Boneh, B. Fisch, A. Gabizon, and Z. Williamson. A simple range proof from polynomial commitments, 2020. link.
3. D. Boneh and V. Shoup. *A graduate course in applied cryptography (version 0.6)*. 2023. cryptobook.us.

4. W. Borgeaud. Membership proofs from polynomial commitments, 2020. https://solvable.group/posts/membership-proofs-from-polynomial-commitments/.
5. B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. Zether: Towards privacy in a smart contract world. In J. Bonneau and N. Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443, Kota Kinabalu, Malaysia, Feb. 10–14, 2020. Springer, Cham, Switzerland.
6. 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)*, pages 315–334. IEEE, 2018.
7. B. Bünz, B. Fisch, and A. Szepieniec. Transparent snarks from DARK compilers. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.
8. R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, and R. D. Rothblum. Fiat-shamir from simpler assumptions. *IACR Cryptol. ePrint Arch.*, page 1004, 2018.
9. D. Catalano and D. Fiore. Vector commitments and their applications. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 55–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
10. B. Chen, B. Bünz, D. Boneh, and Z. Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT (2)*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023.
11. A. Chiesa, M. A. Forbes, and N. Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017.
12. A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020.
13. A. Chiesa and E. Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024.
14. M. Christ, F. Baldimtsi, K. K. Chalkias, D. Maram, A. Roy, and J. Wang. Sok: Zero-knowledge range proofs. In *AFT*, volume 316 of *LIPIcs*, pages 14:1–14:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
15. G. Couteau, D. Goudarzi, M. Klooß, and M. Reichle. Sharp: Short relaxed range proofs. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *ACM CCS 2022*, pages 609–622, Los Angeles, CA, USA, Nov. 7–11, 2022. ACM Press.
16. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany.
17. L. Eagen, D. Fiore, and A. Gabizon. cq: Cached quotients for fast lookups. *IACR Cryptol. ePrint Arch.*, page 1763, 2022.
18. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.
19. A. Gabizon and D. Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. *IACR Cryptol. ePrint Arch.*, page 1447, 2022.
20. A. Gabizon and Z. J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, page 315, 2020.
21. R. Gao, Z. Wan, Y. Hu, and H. Wang. A succinct range proof for polynomial-based vector commitment. In B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, editors, *ACM CCS 2024*, pages 3152–3166, Salt Lake City, UT, USA, Oct. 14–18, 2024. ACM Press.
22. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), Sept. 2015.
23. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.
24. T. Kohrita and P. Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023.
25. C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, Oct. 1992.
26. C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.
27. G. Maxwell. Confidential transactions, 2015.
28. J. Posen and A. A. Kattis. Caulk+: Table-independent lookup arguments. *IACR Cryptol. ePrint Arch.*, page 957, 2022.
29. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, oct 1980.

30. S. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Cham, Switzerland.
31. S. Setty and J. Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.
32. S. T. V. Setty, J. Thaler, and R. S. Wahby. Unlocking the lookup singularity with lasso. In *EUROCRYPT (6)*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2024.
33. M. Stadler. Publicly verifiable secret sharing. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 190–199, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
34. J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.
35. R. S. Wahby, I. Tzialla, a. shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.
36. T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Cham, Switzerland.
37. A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin. Caulk: Lookup arguments in sublinear time. In *CCS*, pages 3121–3134. ACM, 2022.
38. A. Zapico, A. Gabizon, D. Khovratovich, M. Maller, and C. Ràfols. Baloo: Nearly optimal lookup arguments. *IACR Cryptol. ePrint Arch.*, page 1565, 2022.
39. J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876, 2020.
40. R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, page 216–226, Berlin, Heidelberg, 1979. Springer-Verlag.

# A Deferred Preliminaries

## A.1 Public Key Encryption

**Definition 5 (Public Key Encryption [3](Ch. 11)).** *Public key encryption is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pk}, \mathsf{sk}$*, where* $\mathsf{pk}$ *and* $\mathsf{sk}$ *are a pair of public and secret keys respectively*
- $\mathsf{Enc}(\mathsf{pk}, m; r) \to \mathsf{ct}$*, where* $\mathsf{ct}$ *is a ciphertext,* $m$ *is a message, and* $r$ *is randomness.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$

*A public key encryption scheme is additively homomorphic if the following holds:*

$$\mathsf{Enc}(\mathsf{pk}, m; r) \cdot \mathsf{Enc}(\mathsf{pk}, m'; r') = \mathsf{Enc}(\mathsf{pk}, m + m'; r + r').$$

Let $\mathbb{G}$ be a cyclic group of prime order $p$ with generator $g$. Let $q << p$. Below we review the ElGamal encryption scheme where the message space is $\mathbb{Z}_q$.

**Definition 6 (ElGamal encryption [3, Ch. 11.5]).** *ElGamal encryption for a message space* $\mathbb{Z}_q$ *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk} \in \mathbb{G}, \mathsf{sk} \in \mathbb{Z}_p)$
  1. $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$
  2. $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
  3. *Output* $(\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Enc}(\mathsf{pk} \in \mathbb{G}, m \in \mathbb{Z}_q, r \leftarrow\!\!\$\ \mathbb{Z}_p) \to \mathsf{ct} \in \mathbb{G}^2$
  1. $\mathsf{ct}_1 \leftarrow g^r$
  2. $\mathsf{ct}_2 \leftarrow \mathsf{pk}^r \cdot g^m$
  3. *Output* $(\mathsf{ct}_1, \mathsf{ct}_2)$
- $\mathsf{Dec}(\mathsf{sk} \in \mathbb{Z}_p, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) \in \mathbb{G}^2) \to m \in \mathbb{Z}_q$

1. $y \leftarrow \mathsf{ct}_2/\mathsf{ct}_1^{\mathsf{sk}}$
2. *Find* $x \in \mathbb{Z}_q$ *such that* $g^x = y$

The last step of ElGamal decryption involves solving a discrete logarithm problem, so if $q$ is too large, then efficient decryption is not possible.

We additionally note that ElGamal encryption is additively homomorphic. To see why, let $(\mathsf{ct}_1, \mathsf{ct}_2) \leftarrow \mathsf{Enc}(pk, m, r)$, and let $(\mathsf{ct}'_1, \mathsf{ct}'_2) \leftarrow \mathsf{Enc}(pk, m', r')$. Then:

$$(\mathsf{ct}_1 \cdot \mathsf{ct}'_1, \mathsf{ct}_2 \cdot \mathsf{ct}'_2) = (g^{r+r'}, pk^{r+r'} \cdot g^{m+m'}) = \mathsf{Enc}(pk, m+m', r+r')$$

### A.2 Interactive and Non-Interactive Arguments

An interactive argument $\Pi$ may have the following properties:

1. Computational Completeness: $\Pi$ is computationally complete if for every PPT $\mathcal{A}$:

$$\Pr\left[\langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;:\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

   If $\mathcal{A}$ is unbounded, then we say $\Pi$ is perfectly complete.
2. $\delta$-Soundness (adaptive): $\Pi$ is $\delta$-sound for $\delta : \mathbb{N} \to (0,1)$ if for every pair of probabilistic polynomial time adversarial prover algorithm $(\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$, the following holds:

$$\Pr\left[\langle \tilde{\mathcal{P}}_2(\mathbb{i}, \mathbb{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1 \wedge (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) : \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array}\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)$$

   We say a protocol is computationally sound if $\delta$ is negligible. If $\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2$ are unbounded and $\delta$ is negligible, then the protocol is statistically sound. We call a statistically sound protocol a proof and a computationally sound protocol an argument.
3. $\delta$-Knowledge Soundness: $\Pi$ is $\delta$-knowledge sound for $\delta : \mathbb{N} \to (0,1)$ if there exists a PPT oracle machine $\mathcal{E}$ called the extractor such that given oracle access to any pair of PPT adversarial prover algorithms $(\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$, the following holds:

$$\Pr\left[\begin{array}{c} \langle \tilde{\mathcal{P}}_2(\mathbb{i}, \mathbb{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1 \\ \wedge\, (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \end{array} : \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \\ \mathbb{w} \leftarrow \mathcal{E}^{\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2}(\mathsf{gp}, \mathbb{i}, \mathbb{x}) \end{array}\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)$$

   An interactive argument is "knowledge-sound," or simply an "argument of knowledge," if the knowlege error $\delta$ is negligible in $\lambda$. If the adversary is unbounded, then the argument is called an interactive proof of knowledge.
4. Public-coin: $\Pi$ is public-coin if all verifier messages can be computed as a deterministic function given a random public input
5. Honest-Verifier Zero-Knowledge: $\Pi$ is zero-knowledge if there exists a PPT simulator $\mathcal{S}$ such that for every PPT adversary $\mathcal{A}$:

$$\left|\Pr\left[\begin{array}{c} \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) = 1 \\ \wedge\, (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \end{array} : \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array}\right] - \right.$$
$$\left. \geq \Pr\left[\begin{array}{c} \langle \mathcal{S}(\sigma, \mathsf{pp}, \mathbb{x}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1 \\ \wedge\, (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \end{array} : \begin{array}{c} (\mathsf{gp}, \sigma) \leftarrow \mathcal{S}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array}\right]\right| \leq \mathsf{negl}(\lambda)$$

6. Non-interactive: $\Pi$ is non-interactive if it is non-interactive and a proof can be checked by any verifier. Non-interactive proofs may additionally be succinct, which means the proof size and verifier runtime are $o(|w|)$. The verifier can run in linear time in $|x|$. A succinct non-interactive argument of knowledge is called a SNARK.

### A.3 Polynomial Interactive Oracle Proofs

The following definition of PIOPs repeats the definition of algebraic holographic proofs from Marlin [12]. We make two minor generalizations. First, we explicitly allow the instance to contain oracles. This enables us to describe sum-check as a PIOP where the instance contains an oracle to the polynomial over which the sum is being computed. Second, we allow the prover to send vectors of field elements in addition to polynomial oracles. We emphasize that this is just a syntactic change as the prover could easily encode these field elements in a polynomial oracle which the verifier could then query. In our sum-check constructions, we use these field elements to specify polynomials that the verifier can interpolate itself.

**Definition 7 (Public-Coin Polynomial Interactive Oracle Proof [12]).** *A public-coin polynomial interactive oracle proof (PIOP) over a field family $\mathcal{F}$ is a public-coin interactive proof for an indexed oracle relation $\mathcal{R} = (\mathtt{i}, \mathtt{x}, \mathtt{w})$. A public-coin PIOP is specified by a tuple of algorithms $(\mathcal{I}, \mathcal{P}, \mathcal{V})$, polynomial-time computable functions $k, s, d, \ell : \{0,1\}^* \to \mathbb{N}$, and a scalar $\mu \in \mathbb{N}$. $\mathcal{I}, \mathcal{P}, \mathcal{V}$ are the indexer, prover, and verifier respectively. $k$ specifies the number of rounds, $s$ specifies how many $\mu$-variate polynomials are sent in each round, $d$ specifies the degrees bounds of each polynomial, $\ell$ specifies the length of the vector of field elements in each round, $\mu$ specifies the maximum number of variables for each polynomial oracle, and $n$ specifies the number of oracles in the instance. Both the index $\mathtt{i}$ and the instance $\mathtt{x}$ may contain oracles to $\mu$-variate polynomials. $s(0)$ is sum of the number of polynomial oracles in both $\mathtt{i}$ and $\mathtt{x}$. The verifier has oracle access to any such polynomials.*

*In the offline phase, which we call round 0, the indexer $\mathcal{I}$ receives as input a field $\mathbb{F} \in \mathcal{F}$ and an index $\mathtt{i}$ for $\mathcal{R}$, and outputs $s(0) - n$ $\mu$-variate polynomials $f_{0,1}, \ldots, f_{0,s(0)-n}$ where $f_{0,i} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathtt{i}|,0,i)}$.*

*In the online phase, we have an instance $\mathtt{x}$ and witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. Let $\mathcal{O}^{s(0)-n+1}, \ldots, \mathcal{O}^{s(0)}$ be the $n$ oracles that are part of the instance $\mathtt{x}$, and let $f_{0,s(0)-n+1}, \ldots, f_{0,s(0)}$ be the corresponding polynomials where $f_{0,i} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathtt{i}|,0,i)}$. The prover $\mathcal{P}$ receives $(\mathbb{F}, \mathtt{i}, \mathtt{x}, \mathtt{w})$, and the verifier $\mathcal{V}$ receives $(\mathbb{F}, \mathtt{x})$ and oracles $\mathcal{O}^{f_{0,1}}, \ldots, \mathcal{O}^{f_{0,s(0)}}$.*

*The prover $\mathcal{P}$ and verifier $\mathcal{V}$ interact in $k = k(|\mathtt{i}|)$ rounds. In round $i \in [k]$, $\mathcal{V}$ sends a message $\rho_i \in \mathbb{F}^*$ to $\mathcal{P}$, and $\mathcal{P}$ responds with a vector $\boldsymbol{v}_i \in \mathbb{F}^{\ell(|\mathtt{i}|,i)}$ and $s(i)$ $\mu$-variate polynomial oracles $\mathcal{O}^{f_{i,1}}, \ldots, \mathcal{O}^{f_{i,s(i)}}$ where $f_{i,j} \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d(|\mathtt{i}|,i,j)}$. The verifier can query any polynomial it has received any number of times. A query is a vector of field elements $\boldsymbol{x} \in \mathbb{F}^\mu$ for an oracle $\mathcal{O}^{f_{i,j}}$, and the answer is $f_{i,j}(\boldsymbol{x})$. At the end of the interaction, $\mathcal{V}$ outputs accept or reject.*

*As a proof system, a PIOP satisfies perfect completeness and unbounded knowledge soundness with knowledge-error $\delta$. To recover the witness polynomials, an extractor can query the oracle at arbitrary points.*

### A.4 Sigma Protocol for the Pre-Image of a Group Homomorphism

Below we state the definition of sigma protocols from Chapter 19.5.4 of [3]

**Definition 8 (Sigma Protocol).** *A sigma protocol for a relation $\mathcal{R} = \{x, w\}$ is a pair of interactive algorithms $(\mathcal{P}, \mathcal{V})$ where $\mathcal{P}$ takes $(x, w)$ as input, $\mathcal{V}$ takes $x$ as input, and an interaction between $\mathcal{P}$ and $\mathcal{V}$ is structured as follows:*

- *$\mathcal{P}$ computes a message $t$ called the commitment and sends $t$ to $\mathcal{V}$*
- *$\mathcal{V}$ chooses a challenge $c$ from a finite challenge space $\mathcal{C}$ and sends $c$ to $\mathcal{P}$*
- *$\mathcal{P}$ computes a response $z$ to $\mathcal{V}$ and sends $z$ to $\mathcal{V}$*
- *$\mathcal{V}$ outputs accept or reject as a deterministic function of $x$ and the conversation $(t, c, z)$*

*We require that for all $(x, w) \in \mathcal{R}$, the result of $\mathcal{P}(x, w)$ interacting with $\mathcal{V}(x)$ is accept. If $\mathcal{V}$ accepts on $(t, c, z)$, then $(t, c, z)$ is an accepting conversation.*

*A sigma protocol must have the following properties:*

- *Special-soundness: A sigma protocol $(\mathcal{P}, \mathcal{V})$ is special-sound if there is an efficient deterministic algorithm $\mathcal{E}$ called a witness extractor such that whenever $\mathcal{E}$ is given $x$ and two accepting conversations $(t, c, z)$ and $(t, c', z')$ with $c \neq c'$, $\mathcal{E}$ always outputs $w$ such that $(x, w) \in \mathcal{R}$.*
- *Special Honest-Verifier Zero-Knowledge: A sigma protocol $(\mathcal{P}, \mathcal{V})$ is special honest-verifier zero-knowledge if there exists an efficient probabilistic algorithm $\mathcal{S}$ called the simulator that takes as input $(x, c)$ and satisfies the following properties:*
  - *For all inputs $(x, c)$, $\mathcal{S}$ always outputs a pair $(t, z)$ such that $(t, c, z)$ is an accepting conversation for $x$.*
  - *For all $(x, w) \in \mathcal{R}$, if we compute $c \leftarrow\!\!\$\, \mathcal{C}$, $(t, z) \leftarrow\!\!\$\, \mathcal{S}(x, c)$, then $(t, c, z)$ has the same distribution as that of a transcript conversation between $\mathcal{P}(x, w)$ and $\mathcal{V}(x)$.*

Let $\mathbb{H}_1$ and $\mathbb{H}_2$ be two groups of the same order, and let $\psi : \mathbb{H}_1 \to \mathbb{H}_2$ be a group homomorphism. Let $w \in \mathbb{H}_1$ and let $x \in \mathbb{H}_2$. From [3], we have that a prover can create a sigma protocol to prove the following relation: $\mathcal{R}_\Sigma = \{(x, \psi; w) : \psi(w) = x\}$. We refer to Section 19.5.4 of [3] for the construction. The communication in this construction is linear in the size of $x$ and $w$. For some use cases, this can be efficient. For instance, in the $\Pi_{\mathsf{scalar}}$ protocol described at the end of Section 4, elements from $\mathbb{H}_1$ and $\mathbb{H}_2$ have constant size, so communication is constant. However, in Section D.1, we use a sigma protocol to prove consistency between $m$ ciphertexts and a single polynomial commitment, and in this case, the sizes of $\mathbb{H}_1$ and $\mathbb{H}_2$ depend on the number of ciphertexts. If we were to use the standard group homomorphism sigma protocol from [3], then communication would also be linear in the number of ciphertexts.

Attema et al. [1] show how a prover can open multiple group homomorphisms on a single multi-exponentiation with logarithmic communication. This construction is not quite sufficient for our use case because we need the group homomorphism to take as input values that are not exponents in the multiexponentiation. Fortunately, with the generalization from Appendix A in [1] for compressing sigma protocols for arbitrary group homomorphisms, it is straightforward to construct a compressed sigma protocol for our use case with logarithmic communication. That is, let $\mathbb{G}$ be a group of prime order $p$, let $\boldsymbol{v} \in \mathbb{Z}_p^n$, $\boldsymbol{r} \in \mathbb{Z}_p^\ell$, $P, g_1, \ldots, g_n, x_1, \ldots, x_m \in \mathbb{G}$, and let $\psi_1, \ldots, \psi_m$ be homomorphisms from $\mathbb{Z}_p \to \mathbb{G}$. Then we can prove the following relation with $O(\log n)$ communication:

$$R_{\text{AmorHomExt}} = \left\{ \begin{pmatrix} g_1, \ldots, g_n, \\ P, x_1, \ldots, x_m, \\ \psi_1, \ldots, \psi_m \end{pmatrix} ; \boldsymbol{v}, \boldsymbol{r} : \begin{array}{c} P = \prod_{i=1}^n g_i^{v_i}, \\ x_1 = \psi_1(\boldsymbol{v}, \boldsymbol{r}), \ldots, x_m = f_m(\boldsymbol{v}, \boldsymbol{r}) \end{array} \right\}$$

So if we reduce our consistency check to a relation of this form, then we can use the constructions from Attema et al. [1] to create a proof with logarithmic communication. A small technicality is that we need $n + \ell$ to be a power of 2; if this is not the case, then we can pad the witness.

### A.5 Polynomial Commitment Scheme Security Definitions

A polynomial commitment scheme $\mathsf{C}$ is correct if $\forall \lambda, d \in \mathbb{N}, \boldsymbol{x} \in \mathbb{F}^\mu, f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$:

$$\Pr\left[ b = 1 \;:\; \begin{array}{l} \mathsf{pp} \leftarrow\!\!\$\, \mathsf{Setup}(1^\lambda, d) \\ r \leftarrow\!\!\$\, \mathcal{R}_C \\ (\mathsf{com}, r_{\mathsf{hint}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f, r) \\ y \leftarrow f(\boldsymbol{x}) \\ b \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{com}, \boldsymbol{x}, y, d, \mu; f, r) \end{array} \right] = 1$$

A polynomial commitment scheme $\mathsf{C}$ is binding if for all PPT adversaries $\mathcal{A}$ and for all $d \in \mathbb{N}$:

$$\Pr\left[b_0 = b_1 \neq 0 \wedge f_0 \neq f_1 \;:\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (\mathsf{com}, f_0, f_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_0, r_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_1, r_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

A polynomial commitment scheme $\mathsf{C}$ is hiding if for any PPT adversary $\mathcal{A}$ and for all $d \in \mathbb{N}$:

$$\Pr\left[b \neq b' \;:\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (f_0, f_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow\!\!\$\ \{0,1\} \\ r \leftarrow\!\!\$\ \mathcal{R}_C \\ (\mathsf{com}_b, r_b) \leftarrow \mathsf{Commit}(\mathsf{pp}, f_b, r) \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{st}, \mathsf{com}_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

A polynomial commitment scheme can optionally satisfy the following properties:

- $\mathsf{C}$ is additively homomorphic if for all $f, g \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$,

$$\mathsf{Commit}(\mathsf{pp}, f, r_f) \cdot \mathsf{Commit}(\mathsf{pp}, g, r_g) = \mathsf{Commit}(\mathsf{pp}, f + g, r_f + r_g).$$

- $\mathsf{C}$ is zero-knowledge if the $\mathsf{Eval}$ protocol is a zero-knowledge interactive argument of knowledge.
- $\mathsf{C}$ is extractable if for any PPT adversary $\mathcal{A}$ and for all $d \in \mathbb{N}$, there exists a PPT extractor $\mathcal{E}$ such that:

$$\Pr\left[\begin{array}{c} \mathsf{Eval}(\mathsf{pp}, \mathsf{com}, \boldsymbol{x}, y, d, \mu; f, r) = 1 \\ \wedge (\mathsf{com} \neq \mathsf{com}' \vee f(\boldsymbol{x}) \neq y) \end{array} \;:\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ r_{\mathcal{A}} \leftarrow \mathcal{R} \\ H \leftarrow \mathcal{H} \\ (\mathsf{com}, \boldsymbol{x}, y) \leftarrow \mathcal{A}^H(\mathsf{pp}, d; r_{\mathcal{A}}) \\ (f, r) \leftarrow \mathcal{E}^H(\mathsf{pp}, d; r_{\mathcal{A}}) \\ (\mathsf{com}', r'_{\mathsf{hint}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f, r) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We assume that KZG commitments [23] are straight-line extractable, i.e. without rewinding the malicious committer.

**Additively Homomorphic Polynomial Commitment Schemes.** Below we describe the univariate and multivariate homomorphic commitment schemes used in our constructions.

KZG [23] is an additively homomorphic polynomial commitment scheme for univariate polynomials. The variant of KZG described in [24] is perfectly hiding and has knowledge sound evaluation proofs. It is defined for any groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with the same prime order and a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $g_1, g_2, g_T$ be the generators for $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. The setup for hiding KZG outputs $g_1, g_1^\tau, \ldots, g_1^{\tau^d}, g_1^\xi \in \mathbb{G}_1$ and $g_2, g_2^\tau, g_2^\xi \in \mathbb{G}_2$ for randomly sampled $\tau, \xi \leftarrow\!\!\$\ \mathbb{F}$. The commitment to a polynomial $f(X) = a_0 + a_1 X + \ldots a_d X^d$ using randomness $r \leftarrow\!\!\$\ \mathbb{F}$ is $(g_1^\xi)^r g_1^{a_0} \prod_{i \in [d]} (g_1^{\tau^i})^{a_i}$.

Zeromorph [24] is an additively homomorphic zero-knowledge multilinear polynomial commitment scheme that uses the following multilinear-to-univariate transformation $\mathcal{U}_\mu : \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq 1} \to \mathbb{F}[X_1]^{\preceq 2^\mu}$:

$$\mathcal{U}_\mu(f) = \sum_{\boldsymbol{x} \in \{0,1\}^\mu} f(\boldsymbol{x}) (X^{2^0})^{x_1} \cdots (X^{2^{\mu-1}})^{x_\mu}.$$

To commit to a multilinear polynomial $f$, Zeromorph commits to the univariate polynomial $\mathcal{U}_\mu(f)$ using a hiding additively homomorphic univariate polynomial commitment scheme. If we use the perfectly hiding variant of KZG with public parameters $g_1, g_1^\tau, \ldots, g_1^{\tau^{2^\mu - 1}}, g_1^\xi, g_2, g_2^\tau, g_2^\xi$ to instantiate Zeromorph, then a commitment to a $\mu$-variate multilinear polynomial $f$ with randomness $r$ is $(g^\xi)^r \prod_{i \in [2^\mu]} (g^{\tau^{i-1}})^{f(\langle i-1 \rangle)}$.

Note that $\mathcal{U}_\mu(f)$ is a linear isomorphism (i.e., $\mathcal{U}_\mu(f_1) + \mathcal{U}_\mu(f_2) = \mathcal{U}_\mu(f_1 + f_2)$), so since Zeromorph uses an additively homomorphic univariate polynomial commitment scheme as a building block, Zeromorph commitments themselves are also additively homomorphic.

### A.6 Vector Commitment Scheme

A vector commitment scheme allows a prover to commit to a vector $\boldsymbol{v} \in \mathbb{F}^\mu$ and then later produce a proof that the $i^{th}$ element of $\boldsymbol{v}$ is equal to some $x \in \mathbb{F}$.

**Definition 9 (Vector Commitment Scheme [9]).** *A vector commitment scheme for a vector $\boldsymbol{v} \in \mathbb{F}^\mu$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *such that:*

- $\mathsf{Setup}(1^\lambda, \mu) \to \mathsf{pp}$ *where* $\mathsf{pp}$ *are the public parameters to commit to a vector of max length* $\mu$
- $\mathsf{Commit}(\mathsf{pp}, \boldsymbol{v}, r) \to \mathsf{com}$ *where* $\mathsf{com}$ *is a public commitment to* $f$ *using some* $r$ *randomly sampled from a randomness space* $\mathcal{R}_C$
- $\mathsf{Open}(\mathsf{pp}, \mathsf{com}, \boldsymbol{v}, i) \to (\pi, v_i)$ *where* $\pi$ *is an opening proof showing that the vector committed to by* $\mathsf{com}$ *is* $v_i$ *at position* $i$
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, i, v_i) \to \{0, 1\}$ *where an output of* 1 *means that* $v_i$ *is the* $i^{th}$ *element of the committed vector in* $\mathsf{com}$

*A vector commitment scheme* $\mathsf{C}$ *is correct if* $\forall \lambda, d \in \mathbb{N}, \boldsymbol{v} \in \mathbb{F}^\mu, i \in [\mu]$*:*

$$\Pr\left[\begin{array}{l|ll} & \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, d) \\ & r \leftarrow\!\$\ \mathcal{R}_C \\ b = 1 & : \ \mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \boldsymbol{v}, r) \\ & (\pi, v_i) \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, \boldsymbol{v}, i) \\ & b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, i, v_i) \end{array}\right] = 1$$

*A vector commitment scheme* $\mathsf{C}$ *is position binding if for all PPT adversaries* $\mathcal{A}$ *and for all* $\mu \in \mathbb{N}$*:*

$$\Pr\left[b = 1 \ \wedge \ b' = 1 \ \wedge v_i \neq v_i' \ : \ \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (\mathsf{com}, v_i, v_i', i, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, i, v_i) \\ b' \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi', i, v_i') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

### A.7 Sum-Check

In Figure 4, we rewrite the classic sum-check to match our PIOP definition (Definition 3). Because sum-check is a public-coin protocol with non-adaptive queries, we divide our description into an interactive phase and a query phase. In each round in the interactive phase, the prover sends a degree $d$ univariate polynomial. In the query phase, the verifier performs some checks on these univariate polynomials and makes a single query to the oracle to $f$.

### A.8 Borgeaud Range Proof

Let $\mathsf{C}$ be an additively homomorphic zero-knowledge hiding univariate polynomial commitment scheme with commitment space $\mathcal{C}$. Recall that $\mathsf{CorrelatedRandomness}(b, n, s)$, is a function that outputs uniformly random $s_1, \ldots, s_n$ such that $s = \sum_{i \in [n]} s_i \cdot b^{i-1}$. Figure 5 shows an interactive argument between a prover $\mathcal{P}$ and $\mathcal{V}$ for the following relation:

$$\{(\mathsf{pp}; n, \mathsf{com}_f; r_f, z, s) : \mathsf{com}_f = \mathsf{C.Commit}(\mathsf{pp}, sX + z, r_f) \wedge z \in [0, n)\}$$

**Prover's Input:** $H, f$
**Verifier's Input:** $H, \mathcal{O}^f$
where $H \in \mathbb{F}$, $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\preceq d}$, and $\mathcal{O}^f$ is a polynomial oracle to $h$

Interactive Phase

1. $\mathsf{SC.SendPolys}(\mathcal{P}(h), \mathcal{V}) \to \mathcal{P}(\{\rho_i\}_{i=1}^{\mu-1}, \{f_i\}_{i=1}^{\mu}), \mathcal{V}(\{\rho_i\}_{i=1}^{\mu-1}, \{f_i\}_{i=1}^{\mu})$:
   (a) In round 1:
      i. $\mathcal{P}$ sends the following univariate polynomial to $\mathcal{V}$:

$$f_1(X_1) \coloneqq \sum_{b_2, \ldots, b_\mu \in \{0,1\}^{\mu-1}} f(X_1, b_2, \ldots, b_\mu)$$

      ii. $\mathcal{V}$ samples a random challenge $\rho_1 \leftarrow\!\!\$\ \mathbb{F}$ to send to $\mathcal{P}$.
   (b) In round $i \in [2, \mu)$:
      i. $\mathcal{P}$ sends the following univariate polynomial to $\mathcal{V}$:

$$f_i(X_i) \coloneqq \sum_{b_{i+1}, \ldots, b_\mu \in \{0,1\}^{\mu-i}} f(\rho_1, \ldots, \rho_{i-1}, X_i, b_{i+1}, \ldots, b_\mu)$$

      ii. $\mathcal{V}$ samples a random challenge $\rho_i \leftarrow\!\!\$\ \mathbb{F}$ to send to $\mathcal{P}$.
   (c) In round $\mu$:
      i. $\mathcal{P}$ sends the following univariate polynomial to $\mathcal{V}$:

$$f_\mu(X_\mu) \coloneqq f(\rho_1, \ldots, \rho_{\mu-1}, X_\mu)$$

Query Phase

$\mathcal{V}\left(H, \{\rho_i\}_{i=1}^{\mu-1}, \{f_i\}_{i=1}^{\mu}\right)$ runs the following algorithms sequentially and outputs accept if they all output 1:

1. $\mathsf{SC.PolyCheck}(H, \{\rho_i\}_{i=1}^{\mu-1}, \{f_i\}_{i=1}^{\mu}) \to 0/1$
   (a) For all $i \in [\mu]$, check if $f_i$ is a univariate polynomial of degree at most $d$, and output 0 if not.
   (b) Output 0 if $H \neq f_1(0) + f_1(1)$.
   (c) For all $i \in [2, \mu + 1)$, check if $f_i$ is a univariate polynomial of degree at most $d$.output 0 if $f_{i-1}(\rho_{i-1}) \neq f_i(0) + f_i(1)$.
   (d) If no output so far, output 1.
2. $\mathsf{SC.Query}(\mathcal{O}^f, \{\rho_i\}_{i=1}^{\mu-1}, \boldsymbol{v}_\mu) \to 0/1$
   (a) Sample $\rho_\mu \leftarrow\!\!\$\ \mathbb{F}$. Let $\boldsymbol{\rho} \coloneqq (\rho_1, \ldots, \rho_\mu)$
   (b) Query $\mathcal{O}^f$ to obtain $f(\boldsymbol{\rho})$. Output 1 if $f_\mu(\rho_\mu) = f(\boldsymbol{\rho})$ and 0 otherwise.

**Fig. 4.** Sum-Check PIOP

**Prover's Input:** $\mathsf{pp}, \mathsf{com}_f, r_f, z, s$

**Verifier's Input:** $\mathsf{pp}, \mathsf{com}_f$

– $\mathcal{P}$:
  1. Set $z_1, \ldots, z_n \leftarrow \langle z \rangle$
  2. Set $s_1, \ldots, s_n \leftarrow \mathsf{CorrelatedRandomness}(2, \log n, s)$
  3. Set $r_{f_1}, \ldots, r_{f_n} \leftarrow \mathsf{CorrelatedRandomness}(2, \log n, r_f)$
  4. For all $i \in [\log n]$, define $f_i(X) \coloneqq s_i X + z_i$ and compute $\mathsf{com}_{f_i} \leftarrow \mathsf{Commit}(\mathsf{pp}, f_i, r_{f_i})$
  5. For all $i \in [\log n]$, define $h_i(X) \coloneqq \frac{f_i(X)(f_i(X)-1)}{X}$ and compute commitment $\mathsf{com}_{h_i} \leftarrow \mathsf{Commit}(\mathsf{pp}, h_i, r_{h_i})$ for $r_{h_i} \leftarrow\!\!\$\ \mathbb{F}$
  6. Send $\mathsf{com}_{f_1}, \ldots, \mathsf{com}_{f_{\log n}}, \mathsf{com}_{h_1}, \ldots, \mathsf{com}_{h_{\log n}}$ to $\mathcal{V}$

– $\mathcal{V}$:
  1. Check that $\mathsf{com}_f = \prod_{i \in [\log n]} \mathsf{com}_{f_i} \cdot 2^{i-1}$. If not, output reject
  2. Send random $\rho$ to $\mathcal{P}$

– $\mathcal{P}$ and $\mathcal{V}$ perform $\mathsf{Eval}$ protocol to open commitments $\mathsf{com}_{f_1}, \ldots, \mathsf{com}_{f_{\log n}}$, $\mathsf{com}_{h_1}, \ldots, \mathsf{com}_{h_{\log n}}$ at $\rho$ to $y_{f_1}, \ldots, y_{f_{\log n}}, y_{h_1}, \ldots, y_{h_{\log n}}$ respectively. If the output of any of the $\mathsf{Eval}$ protocols is 0, $\mathcal{V}$ rejects.

– For all $i \in [\log n]$, $\mathcal{V}$ checks that $\rho \cdot y_{h_i} = y_{f_i}(y_{f_i} - 1)$. If any check fails, $\mathcal{V}$ outputs reject. Otherwise it outputs accept.

**Fig. 5.** Borgeaud Range Proof

# B Deferred Constructions

## B.1 $\Pi_{\mathsf{scalar}}$ for Zeromorph

To compile DekartProof using Zerormoph [24] as the zero-knowledge polynomial commitment scheme, we need an interactive argument $\Pi^{\mathsf{ZM}}_{\mathsf{scalar}}$ to show that $(\mathsf{pp}_{\mathsf{ZM}}; \mathsf{com}, f) \in \mathcal{L}(\mathcal{R}^{\mathsf{ZM}}_{\mathsf{scalar}})$ where $\mathsf{pp}_{\mathsf{ZM}}$ are the public parameters for Zeromorph, com is a Zeromorph commitment to a multilinear polynomial, and $f$ is a multilinear polynomial. This is equivalent to the prover showing that it knows $(\beta, r)$ such that $\mathsf{com} = \mathsf{ZM.Commit}(\mathsf{p}_{\mathsf{ZM}}, \beta \cdot f, r)$. Let $g^{\xi}, g, \ldots, g^{\tau^{2^{\mu}}}$ be the public parameters of the Zeromorph scheme. A Zeromorph commitment to the public $f$ with randomness 0 can be computed as $\mathsf{com}_f = \prod_{i \in [2^{\mu}]} (g^{\tau^i})^{\cdot f(\langle i-1 \rangle)}$. If the prover is telling the truth, then $\mathsf{com} = (g^{\xi})^r \prod_{i \in [2^{\mu}]} (g^{\tau^i})^{\beta \cdot f(\langle i-1 \rangle)} = (g^{\xi})^r \mathsf{com}_f^{\beta}$. We can prove com is of this form with a sigma protocol for the pre-image of a group homomorphism where $\mathbb{H}_1 \coloneqq \mathbb{Z}_p^2$, $\mathbb{H}_2 \coloneqq \mathbb{G}$, and $\psi(\beta, r) \coloneqq (g^{\xi})^r \mathsf{com}_f^{\beta}$. This is known as Okamoto's protocol (see 19.5.1 in [3]).

## B.2 Proving Values are in Arbitrary Ranges

Say we want to prove that for $m-1$ values $v_1, \ldots, v_{m-1}$, each $v_i \in [\ell_i, h_i)$. In [21], they explain how to do so by invoking the standard MissileProof construction for proving $m-1$ values are in a single range $[0, n)$ twice. We can achieve the same result with DekartProof, and we repeat the construction here for completeness. To prove a value $v_i \in [\ell_i, h_i)$, a prover only needs to show that $v_i - \ell_i \geq 0$ and $h_i - v_i \geq 0$. Assume $m + 1 = 2^{\mu}$. In the DekartProof protocol, we have a polynomial $f = \sum_{i \in [m-1]} v_i \cdot \mathsf{eq}_{\langle i-1 \rangle}$. Consider the polynomials $f_{\ell} = \sum_{i \in [m-1]} \ell_i \cdot \mathsf{eq}_{\langle i-1 \rangle}$ and $f_h = \sum_{i \in [m-1]} h_i \cdot \mathsf{eq}_{\langle i-1 \rangle}$. Proving that $v_i - \ell_i \geq 0$ and $h_i - v_i \geq 0$ for all $i \in [m-1]$ is equivalent to proving $f - f_{\ell}$ and $f_h - f$ are both positive everywhere on the hypercube except for at $(1, \ldots, 1)$. Proving that some polynomial is positive everywhere on the hypercube except for at $(1, \ldots, 1)$ is equivalent to proving that the evaluations of this polynomial everywhere on the hypercube except for at $(1, \ldots, 1)$ are in $[0, 2^b)$ for some sufficiently large $b$, which is exactly the purpose of DekartProof. We can therefore invoke the DekartProof prover twice to prove that a set of values are in arbitrary ranges. The only change we make to the standard DekartProof protocol is that the verifier still receives a blinded evaluation of $f$ at the end of the protocol, and to obtain the blinded evaluation of $f - f_{\ell}$, the verifier computes the evaluation of $f_{\ell}$ (which is public) and subtracts this from the oracle query answer (the blinded evaluation of $f_h - f$ is computed analogously). We can save some field work for the verifier by combining the bit proofs

from both DekartProof invocations, but the prover will still need to send $2\log n$ commitments to multilinear polynomials.

# C Deferred Proofs

## C.1 Round-by-Round Soundness Proof for Theorem 1

We assume that all the extractor are straight-line to avoid issues with rewinding. Let $\mathsf{state}_O$ denote the state function of the HPIOP. Without the loss of generality, we can split $k$ rounds of $\Pi$ into three parts: $k$ rounds corresponding to the HPIOP interactive phase, $k_1$ rounds of the queries of type 1 and $k_2$ rounds of the queries of type 2.

**Step 0. Empty transcript.** For the empty transcript, we set $\mathsf{state}(\mathtt{i}, \mathtt{x}, \emptyset) = \mathsf{state}_O(\mathtt{i}_O, \mathtt{x}_O, \emptyset)$. Note that, according to our construction, $\mathtt{i}$ contains $\mathtt{i}_O$, and $\mathtt{x}$ is the same as $\mathtt{x}_O$ except that the oracles to polynomials are replaced with commitments to these polynomials.

**Step 1.** In the phase corresponding to the interactive phase of the HPIOP, the partial transcript:

$$\mathsf{tr} \coloneqq (v_l, \{\mathsf{com}_{l,j}\}_{j\in[s(i)]})_{l<i},$$

and the verifier sends $\rho_i$. We set $\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}||\rho_i) = 1$ if and only if following conditions hold:

a) For $j \in s[i]$ : $(f_{i,j}, r_{i,j}) \leftarrow \mathcal{E}_{\mathsf{C}}(\mathsf{com}_{i,j})$
b) $\mathsf{state}_O(\mathtt{i}_O, \mathtt{x}_O, \mathsf{tr}_O||\rho_i) = 1$, where $\mathsf{tr}_O$ is the corresponding partial transcript of the HPIOP, and has the form $\mathsf{tr}_O = (v_l, \{f_{l,j}\}_{j\in[s(i)]})_{l<i}$.

We can show that

$$\Pr_{\rho_i}[\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}||\rho_i) = 1|\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}) = 0] \leq \delta_i$$

Suppose $\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}) = 0$, therefore, $\mathsf{state}_O(\mathtt{i}_O, \mathtt{x}_O, \mathsf{tr}_O||\rho_{i-1}) = 0$. Similarly, if $\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}||\rho_i) = 1$, then $\mathsf{state}_O(\mathtt{i}_O, \mathtt{x}_O, \mathsf{tr}_O||\rho_i) = 1$. However, by RBR soundness property of HPIOP, probability that $\mathsf{state}_O$ can switch from 0 to 1 in round $i$ is less than $\delta_i$. Since this probability is always below $\delta_i$, we don't need the extractor to be able to extract at this step.

**Step 2.** In the next phase, which corresponds to the querying phase of the HPIOP, $\mathsf{tr}$ is as follows:

$$\mathsf{tr} \coloneqq ((v_i, \{\mathsf{com}_{i,j}\}_{j\in[s(i)]}, \rho_i)_{i<k}, (\tilde{i}, \tilde{j}, f_i, \pi_i^{\mathsf{scalar}})_{i<l}, \tilde{i}, \tilde{j}, f_i, \pi_i^{\mathsf{scalar}})$$

Note that in every round in this phase, the prover sends proofs $\pi_{\mathsf{scalar}}$ for every $((\tilde{i}, \tilde{j}), f_i) \in Q_2$ and the verifier does not send any messages, therefore,

$$\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}) = \mathsf{state}((v_i, \{\mathsf{com}_{i,j}\}_{j\in[s(i)]}, \rho_i)_{i<k}).$$

**Step 3.** The transcript has the form:

$$\mathsf{tr} \coloneqq ((v_i, \{\mathsf{com}_{i,j}\}_{j\in[s(i)]}, \rho_i)_{i<k}, (\tilde{i}, \tilde{j}, f_i, \pi_i^{\mathsf{scalar}})_{i<k_1}, A_1, (S_i, x_i, \boldsymbol{c}_i, \pi_i^{\mathsf{Eval}})_{i<l<k_2}),$$

Note that in every round in this phase, the prover sends proofs $\pi_{\mathsf{Eval}}$ for every $(S_i, x_i, \boldsymbol{c}_i) \in Q_1$. The verifier does not send any messages, therefore,

$$\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr}) = \mathsf{state}((v_i, \{\mathsf{com}_{i,j}\}_{j\in[s(i)]}, \rho_i)_{i<k}, (\tilde{i}, \tilde{j}, f_i, \pi_i^{\mathsf{scalar}})_{i<k_1}).$$

**Step 4. Verifier's decision** We show that if $\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr} = 0)$, then the verifier rejects. Suppose $\mathsf{state}(\mathtt{i}, \mathtt{x}, \mathsf{tr} = 0)$ but the verifier outputs accept, therefore, all of the following conditions hold:

1. $\mathbb{D}_{\mathcal{V}_O}(\mathbb{F}, \mathtt{x}, A_1, \rho_1, \ldots, \rho_{k+1})$ outputs accept
2. $\mathcal{V}_{\mathsf{scalar}}$ outputs accept for all $\pi_i^{\mathsf{scalar}}$
3. $\mathcal{V}_{\mathsf{Eval}}$ outputs accept for all $\pi_i^{\mathsf{Eval}}$

If $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr} = 0)$, then $\mathsf{state}_{\mathcal{O}}(\mathbb{i}_{\mathcal{O}}, \mathbb{x}_{\mathcal{O}}, \mathsf{tr}_{\mathcal{O}} = 0)$ but $\mathbb{D}_{\mathcal{V}_O}(\mathbb{F}, \mathbb{x}, A_1, \rho_1, \ldots, \rho_{k+1})$ outputs accept. This implies that $A_1$, or $Q_2$, or both were not computed correctly.

First, assume $A_1$ was not computed correctly, there exists $a = A_1[l]$ such that for the corresponding $(S, \boldsymbol{x}, \boldsymbol{c}) \in Q_1$, the following holds:

$$\sum_{j=1}^{|S|} c_j f_{S[j]}(\boldsymbol{x}) \neq a,$$

where $(f_{S[j]}, r_{S[j]}) \leftarrow \mathcal{E}_{\mathsf{C}}(\mathsf{com}_{S[j]})$. Since $\mathcal{V}$ accepts, then $\mathcal{V}_{\mathsf{Eval}}$ outputs accept for all $\pi_i^{\mathsf{Eval}}$, including $\pi_l^{\mathsf{Eval}}$. Then, we can build an adversary $\mathcal{A}_{\mathsf{C}}$ that outputs $(\sum_{i=1}^{|S|} c_i \cdot \mathsf{com}_{S[i]}, \boldsymbol{x}, a_l)$. For such tuple, the extractor outputs $\sum_{j=1}^{|S|} c_j f_{S[j]}(\boldsymbol{x}) \neq a$. However, by the extractability property of the PCS, this happens with probability less than $\delta_{\mathsf{C}}$.

Second, assume $Q_2$ was not computed correctly. This means $Q_2$ contains a tuple $(\tilde{i}, \tilde{j}), f_i)$ such that $f_{\tilde{i},\tilde{j}}$ is not a multiple of $f_i$. However, $\mathcal{V}_{\mathsf{scalar}}$ outputs accept for all $\pi_i^{\mathsf{scalar}}$. Then, we build an adversary $\mathcal{A}_{\mathsf{scalar}}$, which simply outputs $(\mathsf{com}_{\tilde{i},\tilde{j}}, f_i)$. The corresponding extractor $\mathcal{E}_{\mathsf{scalar}}$ fails to output a valid witness for $(\mathsf{com}_{\tilde{i},\tilde{j}}, f_i)$. However, by the knowledge soundness property of $\Pi_{\mathsf{scalar}}$, if $\mathcal{V}_{\mathsf{scalar}}$ accepts, then $\mathcal{E}_{\mathsf{scalar}}$ outputs an invalid witness with a probability less than $\delta_{\mathsf{scalar}}$.

Applying union bound to the previous arguments, we can bound the following probability:

$$\Pr[\mathcal{V}(\mathsf{pp}_{\mathsf{C}}, \mathsf{vp}, \mathbb{x}) = 1 | \mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr} = 0)] \leq |A_1|\delta_{\mathsf{C}} + |Q_2|\delta_{\mathsf{scalar}}.$$

Thus, if $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr} = 0)$, then the verifier accepts with negligible probability. **Extractor.** The extractor $\mathcal{E}$ works as follows:

1. For every $\mathsf{com}_{i,j}$ in the transcript $\mathsf{tr}$, run PCS extractor $\mathcal{E}_{\mathsf{C}}(\mathsf{com}_{i,j})$ to obtain $f_{i,j}$.
2. For every tuple $(\sum_{j=1}^{|S|} c_j \cdot \mathsf{com}_{S[j]}, x, a)$ in the transcript $\mathsf{tr}$, check if the following holds

$$\mathsf{C.Commit}(f_{S[j]}, r_{S[j]}) = \mathsf{com}_{S[j]} \text{ and } \sum_{j=1}^{|S|} c_j \cdot f_{S[j]}(x) = a,$$

   where $f_{S[j]}$'s are polynomials extracted in the previous step.
3. For every execution of $\Pi^{\mathsf{scalar}}$, run extractor $\mathcal{E}_{\mathsf{scalar}}$ to obtain $(\beta_i, r_i)$. Check if $\beta_i$ is such that

$$\mathsf{com}_i = \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, \beta_i \cdot f_i, r_i).$$

4. If at least one of the previous checks doesn't hold, output $\perp$. Otherwise, output $\{f_{i,j}\}_{j \in s[i]}$.

The probability that the extractor outputs an invalid witness but the verifier accepts is exactly:

$$\Pr[\mathcal{V}(\mathsf{pp}_{\mathsf{C}}, \mathsf{vp}, \mathbb{x}) = 1 | \mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr} = 0)] \leq |A_1|\delta_{\mathsf{C}} + |Q_2|\delta_{\mathsf{scalar}}.$$

### C.2 Zero-Knowledge Proof for Theorem 1

Let $\mathcal{S}_O$ be the honest-verifier zero-knowledge simulator for $(\mathcal{I}_O, \mathcal{P}_O, \mathcal{V}_O)$, and let $\mathcal{S}_{\mathsf{C}}$ be the simulator for $\mathsf{C.Eval}$, let $\mathcal{S}_{\mathsf{scalar}}$ be the honest-verifier zero-knowledge simulator for $\Pi_{\mathsf{scalar}}$. We describe how to construct an honest-verifier zero-knowledge simulator $\mathcal{S}_{\Pi} = (\mathsf{Setup}, \mathsf{Prove})$ for $\Pi$ with size bound $N$.

$\mathcal{S}_{\Pi}^{\mathcal{V}}.\mathsf{Setup}(1^{\lambda})$:

1. $D := \max\{d(N, i, j) \mid i \in \{0, 1, \ldots, k(N)\}, j \in \{1, \ldots, s(i)\}\}$
2. $(\mathsf{pp}_{\mathsf{C}}, \sigma_{\mathsf{C}}) \leftarrow \mathcal{S}_{\mathsf{C}}(1^{\lambda})$
3. $(\mathsf{gp}_{\mathsf{scalar}}, \sigma_{\mathsf{scalar}}) \leftarrow \mathcal{S}_{\mathsf{scalar}}(1^{\lambda})$

$\sigma_{\mathsf{C}}$ and $\sigma_{\mathsf{scalar}}$ are additional inputs used by $\mathcal{S}_{\mathsf{C}}$ and $\mathcal{S}_{\mathsf{scalar}}$ respectively when generating simulated transcripts. $\mathsf{gp}_{\mathsf{scalar}}$ is used to generate proving and verifying simulation keys for $\mathcal{S}_{\mathsf{scalar}}$. Since the relation for $\mathsf{C.Eval}$ is binary, there is no indexer, and the proving and verifying simulation keys are both equal to $\mathsf{pp}_{\mathsf{C}}$.

When we prove honest-verifier knowledge, our adversary is $\mathcal{A} = (\mathcal{A}_1, \mathcal{V})$, where $\mathcal{A}_1$ takes $\mathsf{pp}_{\mathsf{C}}$ and $\mathsf{gp}_{\mathsf{scalar}}$ as input and outputs an index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and state $\mathsf{st}$, and $\mathcal{V}$ is the honest verifier. Just as in the compiler's indexer, we can parse $(\mathbb{i}_{\mathsf{scalar}}, \mathbb{i}_O) \leftarrow \mathbb{i}$. Let $(\mathsf{pp}_{\mathsf{scalar}}, \mathsf{vp}_{\mathsf{scalar}}) \leftarrow \Pi_{\mathsf{scalar}}.\mathcal{I}(\mathsf{gp}_{\mathsf{scalar}}, \mathbb{i}_{\mathsf{scalar}})$. $\mathcal{S}_{\Pi}$ receives $(\mathsf{pp}_{\mathsf{C}}, \sigma_{\mathsf{C}}, \mathsf{pp}_{\mathsf{scalar}}, \sigma_{\mathsf{scalar}}, \mathbb{i}, \mathbb{x})$ as input and behaves as follows:

$\mathcal{S}_{\Pi}^{\mathcal{V}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{C}}, \sigma_{\mathsf{C}}, \mathsf{pp}_{\mathsf{scalar}}, \sigma_{\mathsf{scalar}}, \mathbb{i}, \mathbb{x})$:

- For round $i \in [k]$,
  1. Receive message $\rho_i \in \mathbb{F}^*$ from $\mathcal{V}$ and forward it to the HPIOP simulator $\mathcal{S}_O(\mathbb{F}, \mathbb{i}, \mathbb{x})$.
  2. Receive $s(i)$ oracles and vector $\boldsymbol{v}_i$ from the HPIOP simulator $\mathcal{S}_O(\mathbb{F}, \mathbb{i}, \mathbb{x})$.
  3. Let $z$ be the zero polynomial. For all $j \in [s(i)]$, compute commitment $\mathsf{com}_{i,j} \leftarrow \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, z, r_{i,j})$ for $r_{i,j} \leftarrow\!\!\$\ \mathbb{F}$.
  4. Send the verifier $\{\mathsf{com}_{i,j}\}_{j=1}^{s(i)}$ and $\boldsymbol{v}_i$.
- Simulate the query phase as follows:
  1. Receive a message $\rho_{k+1} \in \mathbb{F}$ from $\mathcal{V}$.
  2. Use the honest query algorithm of the HPIOP to compute the query set $Q \coloneqq \mathbb{Q}_{\mathcal{V}_O}(\mathbb{F}, \mathbb{x}, \rho_1, \ldots, \rho_{k+1})$
  3. Forward the query set $Q = (Q_1, Q_2)$ to the HPIOP simulator $\mathcal{S}_O(\mathbb{F}, \mathbb{i}, \mathbb{x})$ to obtain the simulated view, which includes answers to all queries. If $\mathcal{S}_O$ aborts, then also abort.
  4. For all $((i,j), f) \in Q_2$, use $\mathcal{S}_{\mathsf{scalar}}$ with additional input $\sigma_{\mathsf{scalar}}$ to simulate a proof that $(\mathsf{pp}_{\mathsf{C}}, \mathsf{com}_{i,j}, f) \in \mathcal{L}(\mathcal{R}^{\mathsf{C}}_{\mathsf{scalar}})$.
  5. Parse the answers to the Type 1 queries from the simulated view as $A_1$
  6. For all $(S, \boldsymbol{x}, \boldsymbol{c}) \in Q_1$, let $a$ be the corresponding answer in $A_1$.Use $\mathcal{S}_{\mathsf{C}}$ with input additional $\sigma_{\mathsf{C}}$ to simulate an opening proof to show that $(\mathsf{pp}_{\mathsf{C}}, \sum_{i=1}^{|S|} c_i \cdot \mathsf{com}_{S[i]}, \boldsymbol{x}, a) \in \mathcal{L}(\mathcal{R}_{\mathsf{C}})$

The query answers from $\mathcal{S}_{\Pi}$ are identically distributed to the query answers of $\mathcal{S}_O$, which is by definition zero-knowledge. Now we only need to discuss the rest of the information provided by $\mathcal{S}_{\Pi}$: the commitments and evaluations proofs. Because we assume $\mathsf{C}$ is hiding, the commitments reveal no further information, and the zero-knowledge property of the $\mathsf{C.Eval}$ proofs additionally guarantee that these evaluation proofs are simulatable.

### C.3 Round-by-Round Knowledge Soundness of the Composition of Round-by-Round Knowledge Sound Protocols

Here we show that round-by-round arguments can be sequentially composed with other arguments.

**Theorem 5.** *If $\Pi_1 = (\mathcal{I}_1, \mathcal{P}_1, \mathcal{V}_1)$ is a $(\delta_1^1, \ldots, \delta_{k_1}^1)$-round-by-round knowledge sound argument for a relation $\mathcal{R}_1$ and $\Pi_2 = (\mathcal{I}_2, \mathcal{P}_2, \mathcal{V}_2)$ is a $(\delta_1^2, \ldots, \delta_{k_2}^2)$-round-by-round knowledge sound argument for a relation $\mathcal{R}_2$, then $\Pi_\wedge$ is a $(\delta_1^1, \ldots, \delta_{k_1}^1, \delta^1 + \delta_1^2, \ldots, \delta^1 + \delta_{k_2}^2)$-round-by-round knowledge sound argument for the relation $\mathcal{R}_\wedge = \mathcal{R}_1 \wedge \mathcal{R}_2$ and the knowledge soundness error is $(k_2 + 1)\delta^1 + \delta^2$, where $\delta^1$ and $\delta^2$ are the knowledge soundness errors of $\Pi_1$ and $\Pi_2$, respectively.*

*Proof.* Completeness. Completeness is obvious.

Round-by-Round Knowledge soundness. Let $\mathbb{x}_\wedge = (\mathbb{x}_1, \mathbb{x}_2)$ and $\mathbb{w}_\wedge = (\mathbb{w}_1, \mathbb{w}_2)$. We denote the state functions of $\Pi_1$ and $\Pi_2$ as $\mathsf{state}_1$ and $\mathsf{state}_2$, respectively.

**Step 0. Empty transcript.** We set $\mathsf{state}_\wedge = (\mathbb{i}_\wedge, \mathbb{x}_\wedge, \emptyset) = 1$ if and only if $\mathsf{state}_1 = (\mathbb{i}_1, \mathbb{x}_1, \emptyset) = 1$ and $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 1$.

**Step 1. Execution of $\Pi_1$.** The transcript has the form $\mathsf{tr} = ((\pi_i^1, \rho_i)_{i<l}, \pi_l^1)$, where $\pi_i^1$'s denote $\mathcal{P}_1$'s messages in round $i$ and $\rho_i$ is the randomness sent by $\mathcal{V}_1$. We set $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 1$ if and only if both following conditions hold:

1. $\mathsf{state}_1(\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}||\rho_i) = 1$
2. $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 1$

The extractor $\mathcal{E}$ works as follows:

1. Run $\mathcal{E}_1$ to obtain $\mathbb{w}_1$
2. Run $\mathcal{E}_2$ to obtain $\mathbb{w}_2$
3. Output $(\mathbb{w}_1, \mathbb{w}_2)$

Suppose $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$ and

$$\Pr_{\rho_i}[\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 1] > \delta_i^1.$$

We can show that $(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathcal{E}(\mathbb{x}, \mathsf{tr})) \in \mathcal{R}_\wedge$.

By the definition of the state function, $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$ if $\mathsf{state}_1(\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}) = 0$, or $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 0$, or both. Note that if $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 0$, then $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 0$ too. The only way $\mathsf{state}_\wedge$ can switch from 0 to 1 is when $\mathsf{state}_1$ switches from 0 to 1 and $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 1$. By round-by-round knowledge soundness of $\Pi_1$, $\mathcal{E}_1$ in that case outputs an invalid witness $\mathbb{w}_1$ with a probability less than $\delta_i^1$. Since $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \emptyset) = 1$, $\mathcal{E}_2$ outputs a valid witness $\mathbb{w}_2$.

**Step 2. Execution of $\Pi_2$.** The transcript has the form $\mathsf{tr} = (\mathsf{tr}_1, (\pi_i^2, \rho_i)_{i<l}, \pi_l^2)$, where $\mathsf{tr}_1 = (\pi_i^1, \rho_i)_{i<k_1})$ is the transcript of $\Pi_1$, $\pi_i^2$'s denote $\mathcal{P}_2$'s messages in round $i$ and $\rho_i$ is the randomness sent by $\mathcal{V}_2$. We set $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 1$ if and only if both following conditions hold:

1. $\mathsf{state}_2(\mathbb{i}_2, \mathbb{x}_2, \mathsf{tr}_2||\rho_i) = 1$
2. $\mathsf{state}_1(\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}_1) = 1$

The extractor works the same as in Step 1. Suppose $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$ and

$$\Pr_{\rho_i}[\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 1] > \delta_i^2.$$

We can show that $(\mathbb{i}_\wedge, \mathbb{x}, \mathcal{E}(\mathbb{x}, \mathsf{tr})) \in \mathcal{R}_\wedge$.

Again, let us analyze when $\mathsf{state}_\wedge$ can switch from 0 to 1. By the definition, $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$ if either $\mathsf{state}_1(\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}_1) = 0$, or $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \mathsf{tr}) = 0$, or both. Note that if $\mathsf{state}_1 = (\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}_1) = 0$, then $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}||\rho_i) = 0$ too. The only way $\mathsf{state}_\wedge$ can switch from 0 to 1 is when $\mathsf{state}_2$ switches from 0 to 1 and $\mathsf{state}_1 = (\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}_1) = 1$. By round-by-round knowledge soundness of $\Pi_2$, $\mathcal{E}_2$ in that case outputs an invalid witness $\mathbb{w}_2$ with a probability less than $\delta_i^2$. Since $\mathsf{state}_1 = (\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}) = 1$, $\mathcal{E}_1$ outputs an invalid witness $\mathbb{w}_1$ with a probability less than $\delta^1 = \sum_{i=1}^{k_1} \delta_i^1$, which is the knowledge soundness error of $\Pi_1$.

**Verifier's decision.** We need to show that if $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$, then $\mathcal{V}_\wedge$ rejects. Observe that $\mathcal{V}_\wedge$ accepts only if both $\mathcal{V}_1$ and $\mathcal{V}_2$ accept. If $\mathsf{state}_\wedge(\mathbb{i}_\wedge, \mathbb{x}_\wedge, \mathsf{tr}) = 0$, then $\mathsf{state}_1 = (\mathbb{i}_1, \mathbb{x}_1, \mathsf{tr}_1) = 0$, or $\mathsf{state}_2 = (\mathbb{i}_2, \mathbb{x}_2, \mathsf{tr}_2) = 0$, or both. Therefore, by round-by-round knowledge soundness of $\Pi_1$ and $\Pi_2$, $\mathcal{V}_1$, or $\mathcal{V}_2$, or both reject, respectively. Thus, $\mathcal{V}_\wedge$ rejects.

**Knowledge Soundness Error.** To get the knowledge soundness error, we apply the union bound and get:

$$\delta \leq \delta^1 + \sum_{i=1}^{k_2} \delta^1 + \delta_i^2 = (k_2 + 1)\delta^1 + \delta^2.$$

Note that the additional factor of $k_2$ for $\delta^1$ comes from the fact that $\mathcal{E}_1$ can be invoked during the execution of $\Pi_2$ while $\mathcal{E}_2$ is called during the execution of $\Pi_1$ only if $(\mathbb{i}_2, \mathbb{x}_2, \mathbb{w}_2) \in \mathcal{R}_2$.

## D Vector Range Proof for Vector Commitments

In this section, we generically describe how to prove the following relation from the introduction using $\mathsf{DekartProof}$, where $\mathsf{VC}$ is a vector commitment scheme:

$$\mathcal{R}^{\mathsf{VC}} = \left\{ (\mathsf{com}; \boldsymbol{v} \in \mathbb{F}_p^d, r \in \mathbb{F}) \quad : \quad \mathsf{com} \in \mathsf{VC.Commit}(\boldsymbol{v}, r), \quad \boldsymbol{v} \in [0, n]^d \right\}$$

Then in Section D.1, we discuss how to produce this proof if the commitment algorithm is ElGamal public key encryption.

For any vector commitment scheme $\mathsf{VC}$, we can prove $\mathcal{R}^{\mathsf{VC}}$ using $\mathsf{DekartProof}$ and a zero-knowledge multilinear polynomial commitment scheme $\mathsf{C}$. First, the prover constructs a multilinear polynomial $f$ whose evaluations over the boolean hypercube are equal to the elements of the committed vector. The prover uses $\mathsf{C}$ to commit to $f$. Second, the prover and verifier run the compiled $\mathsf{DekartProof}$ protocol from Section 5 on the commitment to $f$. Third, the prover and verifier participate in an interactive argument to show that the evaluations of the committed $f$ over the hypercube are equal to elements of the committed vector. The first two steps are straightforward, but the consistency proof requires an additional protocol. More formally, proving that the evaluations over the boolean hypercube of the polynomial committed to by $\mathsf{com}_f$ are equal to the vector elements committed to in $\mathsf{com}$ is equivalent to describing a zero-knowledge interactive argument for the following relation:

$$\mathcal{R}_{\mathsf{C}}^{\mathsf{VC}} = \left\{ \begin{matrix} \mathsf{pp}_{\mathsf{VC}}, \mathsf{pp}_{\mathsf{C}}; \mathsf{com}_f, \mathsf{com}_v; \\ \boldsymbol{v}, r_v, f, r_f \end{matrix} \; : \; \begin{matrix} \mathsf{com}_f = \mathsf{C.Commit}(\mathsf{pp}_{\mathsf{C}}, f, r_f) \\ \wedge \; \mathsf{com}_v = \mathsf{VC.Commit}(\mathsf{pp}_{\mathsf{VC}}, \boldsymbol{v}, r_v) \\ \wedge \forall i \in [k], f(\langle i-1 \rangle) = v_i \end{matrix} \right\}$$

The exact argument will depend on both the vector commitment scheme $\mathsf{VC}$ and the commitment scheme $\mathsf{C}$.

### D.1 ElGamal Instantiation of Vector Range Proof for Committed Vector

In this section, we briefly discuss how to instantiate the scheme described above when using ElGamal ($\mathsf{EG}$) as the vector commitment scheme and Zeromorph ($\mathsf{ZM}$) as the underlying zero-knowledge multilinear polynomial commitment scheme. To use ElGamal as a vector commitment scheme, we simply encrypt every element of the vector individually. As discussed earlier, we can describe an interactive argument for $\mathcal{R}^{\mathsf{EG}}$ using $\mathsf{DekartProof}$ and an interactive argument for $\mathcal{R}_{\mathsf{ZM}}^{\mathsf{EG}}$. That is, we describe an argument for the following relation:

$$\mathcal{R}_{\mathsf{ZM}}^{\mathsf{EG}} = \left\{ \begin{matrix} \{\mathsf{pk}_i\}_{i=1}^k \mathsf{pp}_{\mathsf{ZM}}; \{\mathsf{ct}_i\}_{i=1}^k, \mathsf{com}_f; \\ \{v_i, r_i\}_{i=1}^k, f, r_f \end{matrix} \; : \; \begin{matrix} \mathsf{com}_f = \mathsf{ZM.Commit}(\mathsf{pp}_{\mathsf{ZM}}, f, r_f) \\ \wedge \; \forall i \in [k], \mathsf{ct}_i = \mathsf{EG.Enc}(\mathsf{pk}_i, v_i; r_i) \\ \wedge \forall i \in [k], f(\langle i-1 \rangle) = v_i \end{matrix} \right\}$$

Let $k+1 = 2^\mu$, and let $g, h$ be a generators of a group $\mathbb{G}$ of prime order $p$ for our ElGamal encryptions. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order $p$ with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $g_1, g_2$ be the generators for $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. The public parameters $\mathsf{pp}_{\mathsf{ZM}}$ of Zeromorph $g_1, g_1^\tau, \ldots, g_1^{\tau^{2^\mu}}, g_1^\xi \in \mathbb{G}_1$ and $g_2, g_2^\tau, g_2^\xi \in \mathbb{G}_2$. We can write each ElGamal encryption $\mathsf{ct}_i = (\mathsf{ct}_i[1], \mathsf{ct}_i[2]) = (g^{r_i}, \mathsf{pk}_i^{r_i} h^{v_i})$, and we can think of the Zeromorph commitment as a multiexponentiation $\mathsf{com}_f = \left( \prod_{i \in [k]} (g_1^{\tau^{i-1}})^{v_i} \right) (g_1^\xi)^r$. Then the interactive argument for $\mathcal{R}_{\mathsf{cnst}}^{\mathsf{EG,ZM}}$ is just an instantiation of the compressed sigma protocol construction from Attema et al. [1] for $R_{\text{AMORHOMEXT}}$ (Section A.4). Here we assume our witness length is a power of 2, but if it is not, we can pad the witness.

Let $\mathsf{DekartProof}^{\mathsf{ZM}}$ be the compilation of $\mathsf{DekartProof}$ that uses Zeromorph as the zero-knowledge polynomial commitment scheme. We describe the zero-knowledge interactive argument of knowledge for $\mathcal{R}^{\mathsf{EG}}$ in Figure 6.

**Theorem 6.** *Figure 6 is a public-coin zero-knowledge interactive argument of knowledge for* $\mathcal{R}_{\mathsf{encrange}}^{\mathsf{EG}}$

*Proof.* Completeness and public-coin properties are obvious.

Round-by-Round Knowledge Soundness follows from the Composition Theorem 5.

**Prover's Input:** $g, h, \mathsf{pp}_{\mathsf{ZM}} = g_1, g_1^{\tau}, \ldots, g_1^{\tau^{2^\mu}}, g_1^{\xi}, g_2, g_2^{\tau}, g_2^{\xi}; \{\mathsf{pk}_i, \mathsf{ct}_i\}_{i=1}^k, n; \{v_i, r_i\}_{i=1}^k$

**Verifier's Input:** $g, h, \mathsf{pp}_{\mathsf{ZM}}; \{\mathsf{pk}_i, \mathsf{ct}_i\}_{i=1}^k, n$

1. $\mathcal{P}$ defines $f \coloneqq \sum_{i=1}^k v_i \cdot \mathsf{eq}_{(\langle i-1 \rangle)}$. $\mathcal{P}$ computes $\mathsf{com}_f = \mathsf{ZM.Commit}(\mathsf{pp}_{\mathsf{ZM}}, f, r_f)$ for some $r_f \leftarrow\!\!\$\, \mathbb{F}$.
2. $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{DekartProof}^{\mathsf{ZM}}$ where $\mathcal{P}$'s input is $(n, k+1, \mathsf{com}_f; f, r_f)$ and $\mathcal{V}$'s input is $(n, k+1, \mathsf{com}_f)$. If the output of this interaction is reject, $\mathcal{V}$ outputs reject.
3. For all $i \in [k]$, let $\psi_i : \mathbb{Z}_p^{k+1} \times \mathbb{Z}_p^k \to \mathbb{G}$ be a group homomorphism such that for all $\boldsymbol{v} \in \mathbb{Z}_p^{k+1}, \boldsymbol{r} \in \mathbb{Z}_p^k$, $\psi_i(\boldsymbol{v}, \boldsymbol{r}) = \mathsf{pk}_i^{r_i} h^{v_i}$.
4. $\mathcal{P}$ and $\mathcal{V}$ engage in a compressed sigma protocol ([1]) to prove $(g_1, g_1^{\tau}, \ldots, g_1^{\tau^{2^\mu}}, g_1^{\xi}, \mathsf{com}_f, \mathsf{ct}_1[2], \ldots, \mathsf{ct}_k[2], \psi_1, \ldots, \psi_k) \in \mathcal{L}(\mathcal{R}_{\text{AMORHOMEXT}})$ where $\mathcal{P}$'s witness is $((v_1, \ldots, v_k, r_f), (r_1, \ldots, r_k))$.

**Fig. 6.** Proving the Decryptions of $k$ ElGamal Ciphertexts are in $[0, n)$

Zero-Knowledge: Let $\mathcal{S}_\Sigma$ be the simulator for the sigma protocol in Step 4 of Figure 6. Let $\mathcal{S}^{\mathsf{ZM}}_{\mathsf{DekartProof}}$ be the HVZK simulator for $\mathsf{DekartProof}^{\mathsf{ZM}}$. We describe a simulator $\mathcal{S}^{\mathcal{V}}$ with oracle access to the honest verifier $\mathcal{V}$ and access to $\mathsf{pp}_{\mathsf{ZM}} = g_1, g_1^{\tau}, \ldots, g_1^{\tau^{2^\mu}}, g_1^{\xi}, g_2, g_2^{\tau}, g_2^{\xi}$ as follows:

$\mathcal{S}^{\mathcal{V}}(g, h, \{\mathsf{pk}_i\}_{i=1}^k \mathsf{pp}_{\mathsf{ZM}}, \{\mathsf{ct}_i\}_{i=1}^k, n)$:

1. Compute $\mathsf{com}_f^* = \mathsf{ZM.Commit}(\mathsf{pp}_{\mathsf{com}}, 0, r_f^*)$ for $r_f^* \leftarrow\!\!\$\, \mathbb{F}$.
2. Use $\mathcal{S}^{\mathsf{ZM}}_{\mathsf{DekartProof}}$ to simulate Step 2 with $(n, k+1, \mathsf{com}_f^*)$ as the instance.
3. For all $i \in [k]$, let $\psi_i^* : \mathbb{Z}_p^{k+1} \times \mathbb{Z}_p^k \to \mathbb{G}$ be a group homomorphism such that for all $\boldsymbol{v} \in \mathbb{Z}_p^{k+1}, \boldsymbol{r} \in \mathbb{Z}_p^k$, $f_i^*(\boldsymbol{v}, \boldsymbol{r}) = \mathsf{pk}_i^{r_i} h^{v_i}$.
4. Run $\mathcal{S}_\Sigma$ on the instance $(g_1, g_1^{\tau}, \ldots, g_1^{\tau^{2^\mu}}, g_1^{\xi}, \mathsf{com}_f^*, \mathsf{ct}_1[2], \ldots, \mathsf{ct}_k[2], \psi_1^*, \ldots, \psi_k^*)$.

Zero-knowledge follows directly from the hiding property of Zeromorph, the zero-knowledge property of $\mathcal{S}^{\mathsf{ZM}}_{\mathsf{DekartProof}}$, and the special HVZK property of $\mathcal{S}_\Sigma$.