Interactive Line-Point Zero-Knowledge with Sublinear Communication and Linear Computation

Fuchun Lin, Chaoping Xing, and Yizhou Yao

Shanghai Jiao Tong University
{linfuchun,xingcp,yaoyizhou0620}@sjtu.edu.cn

Abstract. Studies of vector oblivious linear evaluation (VOLE)-based zero-knowledge (ZK) protocols flourish in recent years. Such ZK protocols feature optimal prover computation and a flexibility for handling arithmetic circuits over arbitrary fields. However, most of them have linear communication, which constitutes a bottleneck for handling large statements in a slow network. The pioneer work AntMan (CCS'22), achieved sublinear communication for the first time within VOLE-based ZK, but lost the advantage of fast proving. In this work, we propose two new VOLE-based ZK constructions that achieve sublinear communication and linear computation, simultaneously. Let $\mathcal{C}$ be a circuit with size S, input size n, and depth d. In particular, our first ZK, specialized for layered circuits, has communication $\mathcal{O}(n + d \log S)$, while our second ZK can be used to prove general circuits and has communication $\mathcal{O}\big(n + d \log S + d^2\big)$. Our results are obtained by introducing the powerful sum-check techniques from the mature line of works on interactive proofs into the context of VOLE-based ZK for the first time. Reminiscent of the *non-interactive* line-point zero-knowledge proof system (ITC'21), we introduce an *interactive line-point zero-knowledge* (ILPZK) proof system, which closely connects with VOLE-based ZK protocols. In addition, our works also enrich the studies of ZK based on interactive proofs, with new interesting features (e.g., having information-theoretic UC-security, naturally supporting any field) achieved.

1 Introduction

A proof system allows a prover to convince a verifier that a given input x belongs to some language $\mathcal{L}$. In the literature, *circuit satisfiability* is a popular **NP** language, where the prover proves to the verifier that a given circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$ is satisfiable (i.e., there exists some witness $\mathbf{w} \in \mathbb{F}^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$). Furthermore, a zero-knowledge (ZK) proof of knowledge system guarantees a valid proof can only be generated by the one who holds the witness $\mathbf{w}$, and it reveals nothing about $\mathbf{w}$ beyond $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

Though studies of ZK proof systems date back to 1980s [20], they have not been brought into real-life applications until past few years ago. There are two mainstream focuses on improving the concrete efficiency of ZK proofs, prover

time, and proof size(which constitutes a bottleneck of verification time). For prover time, the best one can hope for is that generating a proof is as fast as verifying the witness in the clear. For proof size, proofs with proof size as small as possible (e.g., constant) are preferred, so that fast verification for large statements becomes possible. However, achieving both goals simultaneously is generally very challenging, in the sense that compressing the proof size is usually accompanied by consumption of prover's computation resources. To our best knowledge, there are a few existing ZK proof systems that achieve linear prover time[1] and sublinear (in the circuit size) proof size simultaneously. We give a brief overview on them as follows.

ZK with linear prover time and sublinear proof size. The first such proof system falls within the scope of *interactive oracle proofs* (IOP), with a line of works [8,9,34,21]. Brakedown [21], as the state-of-the-art, exploits a tensor structure of linear combinations, and makes use of linear-time encodable codes. Due to the asymptotic nature of known constructions for linear-time encodable codes [18,21], the prover's computational overhead might be relatively large for small or medium-sized statements. Moreover, these works operate over large fields with field size at least $\Omega(|\mathcal{C}|)$, restricting the application scenarios.

The second approach follows the GKR interactive proof (IP) protocol [19], with a line of works [30,33,37,36]. At a high level, these works start with optimizations of GKR that have linear prover time, and incorporate a commitment scheme to achieve zero-knowledge. The prover computation in original GKR is around cubic in the circuit size, dominated by evaluating multi-variate polynomials at multiple points. The computational overhead was finally optimized to constant due to efforts in a series of works [14,29,28,33], through exploiting that these polynomials and points are highly structured. On a separate note, original GKR assumes a layered circuit, in which each gate takes input only from gates in the previous layer. The recent breakthrough [36] extends GKR to general circuits and maintains a linear time prover, by carefully describing the much more complicated relations among layers. We finally summarize the different commitment schemes used in these works[2]. In Hyrax [30], the underlying commitment is Pedersen commitment [25], while in Libra [33] and Virgo++ [36], the authors use a pairing-based polynomial commitment. Therefore, all these three protocols are not post-quantum. Moreover, the use of a pairing-based polynomial commitment not only leads to an inherent trusted setup, but also incurs a bigger computational overhead when the witness size is close to the circuit size.

VOLE-based Zero-Knowledge Proofs. Very recently, a line of works [10,11,12] studying how to efficiently generate pseudorandom correlations between parties gave birth to ZK proofs based on random *vector oblivious linear evaluation* (VOLE) correlations. In general, VOLE-based ZK protocols [31,17,16,5,35,2,23] have fascinating performance on the prover side, where the prover only pays

[1] We say a ZK proof has linear prover time, if the prover's computation is only a constant times larger than that of verifying the witness in the clear.

[2] We remark that Virgo [37] uses a polynomial commitment based on FRI [6], leading to prover computation $\mathcal{O}(|C| + n \log n)$, where n is the input size.

a very small constant computational overhead, and proving can be done in a streaming fashion to reduce memory cost. One typical example is the *line-point zero-knowledge* (LPZK) [17], with only $4\times$ to $7\times$ prover computational overhead. LPZK requires sending one field element per multiplication gate and its followup work [16] reduced the communication complexity to $1/2$ field element per multiplication gate for layered circuits, which is still linear in the circuit size[3]. Another optimization of LPZK proposed the QuickSilver [35] demonstrating an important advantage of VOLE-based ZK protocols compared against other proof systems. That is VOLE-based ZK protocols can be easily adapted to prove arithmetic circuits over small fields (even Boolean circuits) with online communication independent of the security parameter. We refer to a recent survey [4] for more details of the VOLE-based ZK literature.

Most VOLE-based ZK protocols are not succinct, with linear proof size and verifier costs almost the same as the prover. The only one exception is AntMan [32], which achieves sublinear proof size but at the cost of increasing prover time to quasi-linear and relying on an additively homomorphic encryption (AHE) scheme. To our best knowledge, there is no VOLE-based ZK protocol achieving linear prover time and sublinear proof size simultaneously. As previous ZK proofs with such properties are realized under frameworks of succinct proofs (either requires a trusted setup, or assumes strong cryptographic assumptions, or only supports larges fields), we ask the following question:

Can we realize ZK protocols with linear prover time and sublinear proof size in the context of VOLE-based ZK?

1.1 Our Contributions

We bring the powerful sum-check protocol [24] along with multi-linear extensions (MLE) into the study of VOLE-based ZK protocols. This is the first time a non-trivial (compressing) classical proof technique is used in this new paradigm. To distinguish from the conventional VOLE-based ZK protocols, we formulate our protocols under a new framework that we dub *interactive line-point zero-knowledge (ILPZK)*. In an LPZK proof, the prover $\mathcal{P}$ *independently* generates an affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$ in an ℓ-dimensional vector space $\mathbb{F}^\ell$ from the circuit $\mathcal{C}$ and the witness $\mathbf{w}$. Then the verifier $\mathcal{V}$ queries a single point $\mathbf{v}(\Delta) := \mathbf{a} \cdot \Delta + \mathbf{b}$ on this line, which allows $\mathcal{V}$ to check the correctness of the computation of every gate. This "gate-by-gate" nature leads to a barrier of squashing the proof length (i.e., the dimension ℓ) to sublinear.

In our protocols, due to the introduction of interactive sum-check, the affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$ is generated *collectively* by the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, where $\mathcal{V}$'s participation is in the form of providing random challenges in each round of sum-check. The involvement of $\mathcal{V}$ in the generation of the affine line makes it possible for implementing probabilistic checking within $\mathbf{v}(x)$, in the sense that wire values are "compressed" by MLE encodings. This is the reason

[3] For special type of statements, sublinear proof size constructions were reported, for example, conjunctions [5] and low-degree polynomials [35].

why the proof size can be made sublinear in the circuit size and the verifier can be light-weight. We believe that ILPZK (see Definition 2 for a formal description) captures the essence of our new protocols and may have independent interest in its own right.

ILPZK for layered arithmetic circuits. Our ILPZK proof for satisfiability of layered arithmetic circuits achieves linear prover time and strict sublinear proof size $\mathcal{O}(n + d \log S)$, where n is the witness length and S is the circuit size.

Theorem 1 (ILPZK for layered arithmetic circuit satisfiability). *Let $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a layered (log-space uniform) arithmetic verification circuit with depth d and number of gates S. There exists an ILPZK proof that proves the satisfiability of $\mathcal{C}$ with the following features:*

- *The prover runs in time $\mathcal{O}(S)$.*
- *The verifier runs in time $\mathcal{O}(n + n' + d \log S + T)$, where $\mathcal{O}(T)$ is the time for evaluating MLEs, and is sublinear for log-space uniform circuits.*
- *Round complexity is $\mathcal{O}(d \log S)$.*
- *Proof length is $\mathcal{O}(n + d \log S)$.*
- *Soundness error is $\mathcal{O}\left(\frac{d \log S}{|\mathbb{F}|}\right)$.*

ILPZK for generic arithmetic circuits. Our ILPZK construction for general arithmetic circuits achieves linear prover time as well and mostly has sublinear proof size except some extreme cases. While each gate of a generic circuit may take inputs from all previous layers, one can still separate a generic circuit into d "layers" where each gate takes at least one input from the previous layer, and d also refers to the depth of the circuit.

Theorem 2 (ILPZK for generic arithmetic circuit satisfiability). *Let $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a generic arithmetic verification circuit with depth d and number of gates S. There exists an ILPZK proof that proves the satisfiability of $\mathcal{C}$ with the following features:*

- *The prover runs in time $\mathcal{O}(S)$.*
- *The verifier runs in time $\mathcal{O}\big(n + n' + d^2 + d \log S + T\big)$, where $\mathcal{O}(T)$ is the time for evaluating MLEs.*
- *Round complexity is $\mathcal{O}(d \log S)$.*
- *Proof length is $\mathcal{O}\big(n + d \log S + d^2\big)$.*
- *Soundness error is $\mathcal{O}\left(\frac{d \log S}{|\mathbb{F}|}\right)$.*

We remark that here the $\mathcal{O}\big(d^2\big)$ term in the proof length is always upper bounded by $\mathcal{O}(S)$. And only in some extremely bad cases (e.g., a narrow circuit with each layer connected to all its previous layers), the upper bound is reached.

Support circuits over small fields. Both of our ILPZK constructions for layered and generic circuits can be adapted to yield significant savings in proof size when proving circuits over small fields (e.g. Boolean circuits) through the use of subfield VOLE. More concretely, taking the Boolean layered circuits for example, the proof size counted in bits is $\mathcal{O}(n + d \log S \log|\mathbb{F}|)$, shaving off a

$\log|\mathbb{F}| = \mathcal{O}(\kappa)$ factor from the witness length n, where κ is the security parameter. This is a big advantage compared against the non-VOLE-based proof systems, as the usual way adapting such a protocol to work for small fields, is to view the small field computations as computations over its extension field, which not only incurs an overall $\mathcal{O}(\kappa)$ overhead, but also possibly demands attentions to guarantee that computations of circuits are indeed over the small field. On the other hand, our protocols still have significant asymptotic advantage compared to $\mathcal{O}(n + S)$-bit communication of conventional VOLE-based ZK protocols (e.g., QuickSilver [35]).

NIZK from compiling ILPZK with VOLE protocols. Both of our ILPZK constructions for layered and generic circuits indeed satisfy additional properties of *public-coin* and *round-by-round soundness*, which allow us to squash interactions via the Fiat-Shamir transform. We first transform an ILPZK into an interactive VOLE-based ZK in the random VOLE-hybrid model, for which we prove security in UC-framework. Then we show that we can obtain designated verifier NIZK arguments from a pseudorandom correlation generator (PCG) instantiation of random VOLE. In addition, we can obtain publicly verifiable NIZK arguments from the VOLE-in-the-head (VOLEitH) [26,3] technique. In general, the former has smaller computation, while the latter has smaller communication. This allows our constructions to find applications in a wider range of scenarios.

1.2 Technical Overview

We provide more details about how we achieve sublinear communication while maintaining linear prover computation in the context of VOLE-based ZK.

"Gate-by-gate" limitations of LPZK. VOLE-based ZK proofs essentially lie in the scope of "commit-and-prove" paradigm, where VOLE serves as a commitment scheme. A vector $\mathbf{x}$ is committed via VOLE in the sense that, the prover obtains random $\mathbf{M}$ and the verifier obtains random $\mathbf{K}, \Delta$ such that $\mathbf{K} = \mathbf{x} \cdot \Delta + \mathbf{M}$, denoted by $[\mathbf{x}]$. VOLE naturally satisfies a linearly homomorphic property, which is the key to allowing to evaluate circuits underneath VOLE. For instance, given $[x], [y]$ for two values x, y and a scalar a, the commitment $[z]$ is obtained by $\mathcal{P}$ setting $M_z := aM_x + M_y$ and $\mathcal{V}$ setting $K_z := aK_x + K_y$, where $z := ax + y$.

Most conventional VOLE-based ZK protocols follow "gate-by-gate" paradigm. We take LPZK [17] for example, as the works [31,17,16,35] differ slightly on low level details concerning how multiplication gates are verified. The prover first commits to the witness and all the intermediate values individually via VOLE as described above. The linear homomorphism property of VOLE implies that verification of addition gates can be realized for free, in the sense that the output commitment can be locally computed from input commitments. For multiplication gates, the prover needs to provide evidences supporting the claim that each multiplication gate is computed correctly, which is done by appending additional entries to VOLE. It is not hard to see that witness, outputs of multiplication gates, and evidences for multiplications should all be included in the VOLE. This gives an intuition that the proof size has to be linear in the circuit size.

"Layer-by-layer" via sum-check protocol. Suppose for simplicity that we have a layered circuit. If we could commit to a whole layer of intermediate wire values using a small number of entries in a VOLE instance, and check relations layer-by-layer with a cost strictly smaller than verifying each gate, this should be sufficient for breaking the linear proof size barrier. This is in fact how the GKR [19] interactive proof protocol proceeds. We emphasise that no zero-knowledge is required there and the prover can reveal committed values directly to the verifier in plaintext. In a bare-bone sketch, GKR proceeds from output layer to input layer sequentially, and employs a sum-check protocol for the layer-by-layer reduction. In more detail, for each layer, GKR starts with a claim about the values of this layer, applies sum-check, and ends with a claim about the values of the previous layer. The final claim about the input layer is actually a claim about the witness, and in fact a much simpler statement to prove.

Cast in our ILPZK framework, and recall that the prover and the verifier are allowed to collectively generate an affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$, the whole affine line $\mathbf{v}$ then can be intuitively divided into d blocks $\mathbf{v}^{(0)}(x), \mathbf{v}^{(1)}(x), \ldots, \mathbf{v}^{(d-1)}(x)$, each specifying a "layer-by-layer" reduction. For simplicity, we call every $\mathbf{v}^{(i)}$ a sub-line of $\mathbf{v}$. Essentially, each sub-line consists of a part serves as commitments of sum-check messages, and a part that proves in zero-knowledge the commitments are honestly generated. These together convince $\mathcal{V}$ that $\mathcal{P}$ has generated a valid GKR proof that a GKR verifier would accept. As the length of each sub-line is sublinear in the number of gates in the corresponding layer, we indeed obtain an ILPZK with sublinear proof size as desired. The generic circuit case is similarly handled incorporating the recent breakthrough results from [36].

Linear time prover. The techniques for achieving a linear time prover in our ILPZK constructions for layered circuits and generic circuits follow closely from Libra [33] and Virgo++ [36], respectively. In addition to costs of running GKR in the clear, extra prover computational costs essentially come from proving all commitments of sum-check messages are correctly generated, which is linear (a small constant multiplicative overhead) in the number of sum-check messages. Therefore, our constructions maintain a linear time prover.

Support arithmetic circuits over any field. We next sketch how our ILPZK constructions can benefit from utilizing subfield VOLE. In a subfield VOLE instance $\mathbf{K} = \mathbf{x} \cdot \Delta + \mathbf{M}$, $\mathbf{x}$ is over a small field $\mathbb{F}_p$, while $\mathbf{K}, \mathbf{M}, \Delta$ are over a large extension field $\mathbb{F}_{p^r}$. In fact, subfield VOLE can be viewed as a commitment scheme for elements of a subfield $\mathbb{F}_p$. Directly replacing the random VOLE in VOLE-based ZK protocols (e.g., QuickSilver [35]) with a random subfield VOLE allows to prove circuits over $\mathbb{F}_p$, with the same asymptotic communication but counted in the number of $\mathbb{F}_p$ elements. However, for the affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$ in our ILPZK protocols, due to the use of MLEs in sum-check, part of $\mathbf{a}$ entries are $\mathbb{F}_{p^r}$ elements even though the arithmetic circuit is over $\mathbb{F}_p$. Intuitively, in order to apply the subfield VOLE techniques, we need an efficient construction of a mixture of standard VOLE and subfield VOLE, where they share the same random Δ. This seems a rather interesting variant of VOLE and might be useful in other application scenarios. We observe that to construct the desired variant of

VOLE, it suffices to show that standard VOLE can be constructed from subfield VOLE. We provide a natural construction by fixing a basis of $\mathbb{F}_{p^r}$ over $\mathbb{F}_p$.

1.3 Comparison with related works

On the one hand, our constructions are within the context of VOLE-based ZK. On the other hand, as we distill ideas from the IP literature, our constructions essentially follow the insightful idea of Cramer and Damgård [15], where they show how to obtain zero-knowledge arguments from IPs by using a cryptographic commitment scheme. We now briefly compare our techniques with related techniques from the literature and carefully argue the pros and cons.

Existing ZK protocols with linear prover time and sublinear proof size. As techniques used in IOP-based ZK protocols are quite different, we omit the detailed comparison. Compared to the state-of-the-art Brakedown [21], our constructions have lower computational overhead for small and medium-sized circuits, and have no restriction on the field size.

Essentially, our constructions share high-level similarities with ZK protocols based on sum-check, e.g., Hyrax [30], Libra [33], Virgo [37], Virgo++ [36], Spartan [27], and Cerberus [22]. In a high-level, the main difference is that here we use a lightweight linearly homomorphic commitment scheme (from VOLE) instead of a heavy one (Pedersen commitment [25]) in Hyrax, or a more powerful polynomial commitment based on FRI [6] in Virgo, or a pairing-based polynomial commitment in Libra and Virgo++, or a discrete-logarithm-based polynomial commitment in Spartan[4], or a Ligero-based polynomial commitment [1] in Cerberus[5]. This brings us the following features.

In addition to standalone-security, our constructions are statistically UC-secure in the random VOLE-hybrid model. As random VOLE correlations can be efficiently generated either from learning parity with noise (LPN) assumption [7] or assuming a pseudo-random generator (PRG), our ZK protocols are post-quantum when implemented. Finally, our constructions natively support proving statements over arbitrary-sized fields, by using the subfield VOLE technique. To our best knowledge, no previous protocol has achieved all these features simultaneously in the context of sum-check-based ZK.

For circuits over sufficiently large fields, our constructions have almost the same asymptotic performance as Libra and Virgo++. Our constructions instead use a light-weight VOLE-based commitment scheme. A consequence is that, when the circuit size is not significantly larger than the witness size, our constructions have slightly larger proof size, while they have larger computational overhead. For Boolean circuits, we offer much better concrete efficiency than Libra and Virgo++ in both communication and computation.

VOLE-based ZK. Wolverine [31] was the first VOLE-based ZK that works for arbitrary-sized fields with communication of 4 field elements per multiplication

[4] Spartan actually has prover computation $\mathcal{O}_\kappa(|\mathcal{C}|)$.

[5] Cerberus can achieve linear prover time by using linear-time encodable codes, while they implement with Reed-Solomon codes, leading to a quasi-linear time prover.

gate. The work LPZK [17] and its follow-up work QuickSilver [35] reduced the communication to 1 field element per multiplication. Next, improved LPZK [16] showed that the communication can be further reduced by half when considering layered circuits. All these works proceed an arithmetic circuit in a "gate-by-gate" flavor, from which they gain benefits of linear prover time and small memory. However, this also incurs proof size inherently linear in the circuit size. Compared to these works, our constructions instead have a "layer-by-layer" flavor, and in general have significantly smaller communication, at the cost of increasing prover computation up to roughly $2\times$, memory increased to $\mathcal{O}(|\mathcal{C}|)$, and round complexity increased to $\mathcal{O}(d \log |\mathcal{C}|)$.

Several works consider optimizing communication complexity in special cases. Mac'n'Cheese [5] focused on proving the disjunction of statements, with communication cost only proportional to the longest one. QuickSilver [35] also proposed a VOLE-based ZK for proving the computation of multiple polynomials, with communication cost linear to the highest degree of these polynomials. AntMan [32] started with a construction that allows for simultaneously proving B evaluations of a circuit $\mathcal{C}$, with communication of $\mathcal{O}(B + |\mathcal{C}|)$ field elements, and prover computation of $\mathcal{O}(B|\mathcal{C}| \log B)$. Then, they showed how to turn this construction into a VOLE-based ZK for a general circuit case, which has communication complexity $\mathcal{O}\big(|\mathcal{C}|^{3/4}\big)$, and prover computation $\mathcal{O}(|\mathcal{C}| \log |\mathcal{C}|)$.

Compared to AntMan (for general circuits), our construction has linear prover time, smaller communication in most application scenarios, the same asymptotic memory consumption, and more rounds. Besides, our construction is information-theoretic in the random VOLE-hybrid model and is public-coin, while AntMan relies on an additively-homomorphic encryption scheme and is not public-coin.

2 Preliminaries

Notations. In this paper, bold letters (e.g. $\mathbf{a}, \mathbf{b}, \mathbf{M}, \mathbf{K}$) are used to denote vectors. Besides, we use x_i to denote the i_{th}-component of the vector $\mathbf{x}$. We use $[a, b]$ (or $[a, b+1)$ sometimes) to denote the set of integers in the range from a to b. A commitment of some value x is denoted by $[x]$. We use $x \xleftarrow{\$} \mathbb{F}$ to denote that x is uniformly sampled from a field $\mathbb{F}$. We identify the set $\{0,1\}^\ell$ and the set $[0, 2^\ell)$ through the natural bijection between the two sets.

Security Model and Functionalities. We provide security proofs of our ZK protocols in the universal composability (UC) framework [13]. In particular, we consider active adversary and static corruption. More details can be found in Appendix A. We formally define a zero-knowledge functionality $\mathcal{F}_{\text{ZK}}$ in Figure 5.

2.1 VOLE-based Commitment

Random vector oblivious linear evaluation (VOLE) is a functionality that allows two parties $P_{\mathcal{S}}, P_{\mathcal{R}}$ to obtain random correlated values. In more detail, the sender $P_{\mathcal{S}}$ obtains two vectors $\mathbf{M}, \mathbf{x}$, while the receiver $P_{\mathcal{R}}$ obtains a scalar Δ and a

vector $\mathbf{K}$ such that $\mathbf{K} = \mathbf{M} + \mathbf{x} \cdot \Delta$. We formalize the ideal functionality of random VOLE over finite field $\mathbb{F}$ in Figure 1.

The above VOLE correlation naturally induces a commitment scheme over $\mathbb{F}$. In the commit phase, the values $\mathbf{x}$ are committed in the sense that through a VOLE functionality the sender obtains $\mathbf{M}$, and the receiver obtains $\Delta, \mathbf{K}$, such that $\mathbf{K} = \mathbf{M} + \mathbf{x} \cdot \Delta$, denoted by $[\mathbf{x}]$. In the unveil phase, $[\mathbf{x}]$ are opened by the sender sending $\mathbf{x}, \mathbf{M}$ to the receiver, who then checks $\mathbf{K} = \mathbf{M} + \mathbf{x} \cdot \Delta$. Such VOLE-based commitment schemes satisfy perfect hiding and statistical binding. Intuitively, the receiver learns nothing about $\mathbf{x}$ before the unveil phase and the sender cannot open $[\mathbf{x}]$ to $\mathbf{x}' \neq \mathbf{x}$ unless he succeeds in guessing the Δ of the receiver.

It can be observed that the above commitment scheme also satisfies a linearly-homomorphic property. Given commitments $[x_1], \ldots, [x_\ell]$ and public coefficients $c, c_1, \ldots, c_\ell \in \mathbb{F}$, the two parties can locally compute $[y] = c + \sum_{i \in [\ell]} c_i \cdot [x_i]$ by setting $y = c + \sum_{i \in [\ell]} c_i \cdot x_i$, $M_y = \sum_{i \in [\ell]} c_i \cdot M_{x_i}$, and $K_y = \Delta \cdot c + \sum_{i \in [\ell]} c_i \cdot K_{x_i}$. In particular, we have $[y] = [x] + (y - x)$. This allows the sender to commit some y of his choice by sending $y - x$ to the receiver, given a commitment $[x]$ of a random x produced by a random VOLE functionality. From this observation, we also define a chosen-input VOLE functionality in Figure 6, which can be easily realized by a random VOLE functionality.

Functionality $\mathcal{F}_{\mathrm{rVOLE}}^{\mathbb{F}}$

Init: Upon receiving (`Init`) from both parties, sample $\Delta \xleftarrow{\$} \mathbb{F}$ if $P_{\mathcal{R}}$ is honest, and receive $\Delta \in \mathbb{F}$ from the adversary $\mathcal{A}$ otherwise. Store Δ and send it to $P_{\mathcal{R}}$. All further (`Init`) commands will be ignored.

Extend: Upon receiving $(\texttt{Extend}, N)$ from both parties, proceed as follows:

1. If $P_{\mathcal{R}}$ is honest, sample $\mathbf{K} \xleftarrow{\$} \mathbb{F}^N$. Otherwise receive $\mathbf{K}$ from $\mathcal{A}$.
2. If $P_{\mathcal{S}}$ is honest, sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}^N$ and compute $\mathbf{M} := \mathbf{K} - \Delta \cdot \mathbf{x} \in \mathbb{F}^N$. Otherwise, receive $\mathbf{x} \in \mathbb{F}^N$ and $\mathbf{M} \in \mathbb{F}^N$ from $\mathcal{A}$ and then recompute $\mathbf{K} := \mathbf{M} + \Delta \cdot \mathbf{x}$.
3. Send $(\mathbf{x}, \mathbf{M})$ to $P_{\mathcal{S}}$ and $\mathbf{K}$ to $P_{\mathcal{R}}$.

Fig. 1: Ideal functionality for random VOLE over $\mathbb{F}$.

2.2 Multi-Linear Extension

The multi-linear extension (MLE) plays a crucial role in the study of interactive proofs. We give a formal definition as follows:

Definition 1 (Multi-Linear Extension). *Let $f : \{0,1\}^\ell \to \mathbb{F}$ be a function that maps the ℓ-dimensional binary hypercube to a field $\mathbb{F}$. The multi-linear*

extension of f is the unique polynomial $\widetilde{f} : \mathbb{F}^\ell \to \mathbb{F}$ such that $\widetilde{f}(x_1, \ldots, x_\ell) = f(x_1, \ldots, x_\ell)$ for all $x_1, \ldots, x_\ell \in \{0,1\}$, where the degree of $\widetilde{f}$ in each variable is 1. Moreover, $\widetilde{f}$ has the form

$$\widetilde{f}(x_1, \ldots, x_\ell) = \sum_{\omega \in \{0,1\}^\ell} f(\omega) \cdot \boldsymbol{\chi}_\omega(x_1, \ldots, x_\ell),$$

where, for any $\omega = (\omega_1, \ldots, \omega_\ell)$,

$$\boldsymbol{\chi}_\omega(x_1, \ldots, x_\ell) := \prod_{i=1}^{\ell} (x_i\omega_i + (1 - x_i)(1 - \omega_i)).$$

The set $\{\boldsymbol{\chi}_\omega : \mathbb{F}^\ell \to \mathbb{F}\}_{\omega \in \{0,1\}^\ell}$ is referred to as the set of multi-linear Lagrange basis polynomials with interpolating set $\{0,1\}^\ell$.

W.l.o.g., assume n is a power of two, then a vector $\mathbf{W} := (w_0, \ldots, w_{n-1})$ over $\mathbb{F}$ can be naturally viewed as a function $W : \{0,1\}^{\log n} \to \mathbb{F}$ such that $W(i) = w_i$ for all $i \in [0, n)$. Hence, we define the multi-linear extension of a vector $\mathbf{W}$ in this way, similarly denoted by $\widetilde{W}$. To evaluate the multi-linear extension $\widetilde{W}$ of $\mathbf{W}$ efficiently, we employ the algorithm proposed in [29], which takes $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ memory usage.

Lemma 1 ([29]). *Assume $n = 2^\ell$ and given $\mathbf{W} \in \mathbb{F}^n$ and $\boldsymbol{r} \in \mathbb{F}^\ell$, one can compute $\widetilde{W}(\boldsymbol{r})$ in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

2.3 Sum-check Protocol and GKR protocol

Our ILPZK proof systems distill ideas from the well-known GKR protocol [19], which involves a multi-variate sum-check protocol [24]. We overview the two protocols here, and detailed descriptions of sum-check can be found in Appendix A.

Sum-check protocols are used to sum up polynomial evaluations on a specific set in a verifiable way, and play a crucial role in designing succinct arguments. We focus on multi-variate sum-check problems, which refer to, given some public ℓ-variate polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ such that

$$H = \sum_{b_1, \ldots, b_\ell \in \{0,1\}} f(b_1, \ldots, b_\ell),$$

without $\mathcal{V}$ computing H by evaluating f at 2^ℓ points on her own. Assuming the maximum degree of f in each variable is d, the sum-check protocol has communication complexity $\mathcal{O}(d\ell)$, round complexity ℓ, and soundness error $\mathcal{O}(d\ell/|\mathbb{F}|)$. Moreover, if one uses a linear time algorithm of evaluating MLEs (e.g., Lemma 1), sum-check can be realized with linear prover time as shown in [28].

Lemma 2 ([28]). *Assume $n = 2^\ell$ and given ℓ-variate multi-linear polynomials $f_1, \ldots, f_d : \mathbb{F}^\ell \to \mathbb{F}$. Applying sum-check on the ℓ-variate polynomial $g := f_1 \cdots f_d$ takes prover time $\mathcal{O}(dn)$.*

The GKR protocol is an interactive proof protocol, and can be used for evaluating layered arithmetic circuits in a verifiable way. More specifically, given a layered circuit $\mathcal{C}$ and inputs $\mathbf{w}$ known to both parties, through invoking GKR, $\mathcal{P}$ can convince $\mathcal{V}$ that h is indeed the evaluation of $\mathcal{C}$ on $\mathbf{w}$ without $\mathcal{V}$ computing $\mathcal{C}(\mathbf{w})$ by herself. In a high level, GKR proceeds the circuit from output to input, in a layer-by-layer fashion. For each layer, GKR starts with a claim about values in this layer and employs a multi-variate sum-check protocol, which reduces the claim to claims about the previous layer. Then to proceed the previous layer, a *condensing* technique was designed in GKR that allows to combine multiple claims to one claim. The final claim about the input layer can be checked by $\mathcal{V}$ directly. Assuming the layered circuit $\mathcal{C}$ has depth d and S gates, GKR has communication complexity $\mathcal{O}(d \log S)$, round complexity $\mathcal{O}(d \log S)$, and soundness error $\mathcal{O}(d \log S/|\mathbb{F}|)$. In [33], GKR is optimized to have a linear prover time. Furthermore, if $\mathcal{C}$ is log-space uniform, authors of [19] showed that the GKR verifier can run in time $\mathcal{O}(\log S)$.

2.4 LPZK [17] and QuickSilver [35]

The LPZK proof system essentially follows the "commit-and-prove" paradigm. Intuitively, the proof consists of two parts, where the first part serves as (linearly-homomorphic) commitments of the extended witness (i.e., wire values in the case of circuit satisfiability), and the second part proves (in zero-knowledge) that values underneath the commitments are exactly the extended witness. In the case of proving arithmetic circuit satisfiability, the extended witness consists of all wire values and it suffices to prove the satisfiability of a degree-2 relation dependent on the circuit topology. To start with, we show two simple examples of LPZK proof [17], which together allow to prove arbitrary degree-2 constraints.

1. ***Linear constraint***: Let $\alpha, \beta \in \mathbb{F}$ be two coefficients known to each other, and suppose $\mathcal{P}$ wants to convince $\mathcal{V}$ that he holds some $a_1, a_2 \in \mathbb{F}$ such that $a_1 = \alpha \cdot a_2 + \beta$. The corresponding LPZK works as follows:
   - $\mathcal{P}$ constructs an affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$ of dimension-3 with $v_1(x) := a_1 \cdot x + b_1$, $v_2(x) := a_2 \cdot x + b_2$ (as commitments of $\mathbf{a}$), and $v_3(x) := 0 \cdot x + b_3$, where $b_1, b_2 \xleftarrow{\$} \mathbb{F}$ and $b_3 = b_1 - \alpha \cdot b_2$.
   - Given an evaluation $\mathbf{v}(\Delta)$, $\mathcal{V}$ checks that $v_3(\Delta) \stackrel{?}{=} v_1(\Delta) - \alpha \cdot v_2(\Delta) - \beta \cdot \Delta$[6].
2. ***Multiplicative constraint:*** Suppose $\mathcal{P}$ wants to convince $\mathcal{V}$ that he holds some $a_1, a_2, a_3 \in \mathbb{F}$ such that $a_3 = a_1 \cdot a_2$. The corresponding LPZK is as follows:
   - $\mathcal{P}$ constructs an affine line $\mathbf{v}(x) := \mathbf{a} \cdot x + \mathbf{b}$ of dimension-4 with $v_1(x) := a_1 \cdot x + b_1$, $v_2(x) := a_2 \cdot x + b_2$, $v_3(x) := a_3 \cdot x + b_3$ (as commitments of $\mathbf{a}$), and $v_4(x) := a_4 \cdot x + b_4$, where $b_1, b_2, b_3 \xleftarrow{\$} \mathbb{F}$, $a_4 := a_1 b_2 + a_2 b_1 - b_3$, and $b_4 := b_1 b_2$.
   - Given an evaluation $\mathbf{v}(\Delta)$, $\mathcal{V}$ checks that $v_1(\Delta) \cdot v_2(\Delta) \stackrel{?}{=} v_3(\Delta) \cdot \Delta + v_4(\Delta)$.

[6] We remark that a_3 must be 0, which can be guaranteed by $\mathcal{P}$ sending b_3 to $\mathcal{V}$ in the clear, which also shortens the VOLE length.

As any degree-2 constraint can be represented by combinations of linear constraints and multiplicative constraints, one can naturally extend the above two constructions and give a construction that proves arbitrary degree-2 relations. In addition, the completeness and security directly follow those of the underlying two simple constructions, on which we defer a detailed discussion to Appendix A.

In this paper, we mainly focus on degree-2 relations consisting of individual t_1 linear constraints and t_2 multiplicative constraints with n inputs. For proving such degree-2 relations, the above LPZK construction requires the affine line $\mathbf{v}(x)$ to have length at least $n + t_1 + t_2$.

Batch-Check Linear/Multiplicative Constraints. Let $\mathcal{C}$ be an arithmetic circuit over $\mathbb{F}$ with n inputs, t multiplication gates, and arbitrary addition gates. For proving the satisfiability of $\mathcal{C}$, both LPZK [17] and QuickSilver [35] first transform it into a degree-2 relation consisting of one linear constraint and t multiplicative constraints with $n + t$ inputs. Thus, the length in LPZK is $n + 2t + 1$ ($n + t$ for "committing", and $t + 1$ for "proving"). QuickSilver is essentially a two-round ILPZK construction, which reduces the length to $n+t+1$ by introducing one additional round of interaction. Instead of building one sub-line per multiplicative constraint in LPZK, QuickSilver builds a sub-line that is the random linear combination of these t sub-lines. Similarly, the random linear combination technique also works for compressing linear constraints. Therefore, for proving a degree-2 relation consisting of individual t_1 linear constraints and t_2 multiplicative constraints with n inputs, it suffices to construct an affine line $\mathbf{v}(x)$ of length $n + 2$ in the ILPZK setting.

3 Interactive Line-Point Zero-Knowledge Proof

In this section, we first give a formal definition of our new notion of Interactive LPZK proof system. Then we show how to compile such a proof system into a publicly verifiable NIZK argument via the VOLE-in-the-Head technique.

3.1 Defining ILPZK

We define interactive LPZK proof systems for arithmetic circuit satisfiability. The interactive LPZK proof system generalizes the original LPZK proof system, in the sense that an LPZK is essentially a 1-round interactive LPZK.

Definition 2 (ILPZK). *A t-round interactive line-point zero-knowledge (ILPZK) proof system for arithmetic circuit satisfiability over $\mathbb{F}$ is given by a pair of algorithms $(\mathbb{P}, \mathbb{V})$ with the following syntax:*

- $\mathbb{P}(\mathcal{C}, \mathbf{w}, i, r_i)$ *is a* PPT *algorithm that given an arithmetic verification circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$, a witness $\mathbf{w} \in \mathbb{F}^n$, a round index $i \in [1, t]$ and a sequence of strings r_i with each r_i being a prefix of r_{i+1}, outputs a pair of vectors $\mathbf{a}^{(i)}, \mathbf{b}^{(i)} \in \mathbb{F}^{\ell_i}$. Let $\ell := \sum_{i=1}^{t} \ell_i$ and $\mathbf{a} := (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(t)}), \mathbf{b} := (\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(t)}) \in \mathbb{F}^{\ell}$, which specify an affine line $\mathbf{v}(x) = \mathbf{a} \cdot x + \mathbf{b}$ of dimension ℓ.*

– $\mathbb{V}(\mathcal{C}, r_t, \Delta, \mathbf{v}_\Delta)$ *is a polynomial time algorithm that, given a string* r_t *and an evaluation* $\mathbf{v}_\Delta$ *of the line* $\mathbf{v}(x)$ *at some point* $\Delta \in \mathbb{F}$*, outputs* `accept` *or* `reject`*.*

With the above ILPZK algorithms, we formally define the ILPZK protocol.

ILPZK Protocol. Given a t-round ILPZK proof system with algorithms $(\mathbb{P}, \mathbb{V})$ as defined in Def. 2, and a chosen-input VOLE functionality $\mathcal{F}_{\text{VOLE}}$ in Figure 6, there exists a t-round interactive protocol $\Pi(\mathbb{P}, \mathbb{V})$ proceeding as follows:

1. At the beginning, the verifier $\mathcal{V}$ picks a $\Delta \in \mathbb{F}$, and sends it to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$. Let r_0 be an empty string.
2. In each round i, where $i = 1, 2, \ldots, t$,
   - $\mathcal{V}$ picks a string s_i and sends it to the prover $\mathcal{P}$. Then $\mathcal{P}$ and $\mathcal{V}$ compute r_i by appending s_i to the end of r_{i-1}.
   - $\mathcal{P}$ runs $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}) \leftarrow \mathbb{P}(\mathcal{C}, \mathbf{w}, i, r_i)$. Then $\mathcal{P}$ sends $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$, which returns $\mathbf{v}_\Delta^{(i)} := \mathbf{a}^{(i)} \cdot \Delta + \mathbf{b}^{(i)}$[7] to $\mathcal{V}$.
3. After t-round of interactions, $\mathcal{V}$ already obtains $\mathbf{v}_\Delta := (\mathbf{v}_\Delta^{(1)}, \ldots, \mathbf{v}_\Delta^{(t)})$. Finally, $\mathcal{V}$ outputs $\mathbb{V}(\mathcal{C}, r_t, \Delta, \mathbf{v}_\Delta)$.

The ILPZK algorithms $(\mathbb{P}, \mathbb{V})$ should at least satisfy the following:

- **Perfect Completeness.** For any arithmetic circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$ and witness $\mathbf{w} \in \mathbb{F}^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$, the verifier $\mathcal{V}$ in $\Pi(\mathbb{P}, \mathbb{V})$ outputs `accept` with probability 1, if the prover $\mathcal{P}$ honestly follows $\Pi(\mathbb{P}, \mathbb{V})$.
- ε**-Soundness.** For every arithmetic circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$ such that $\mathcal{C}(\mathbf{w}) \neq \mathbf{1}$ for all $\mathbf{w} \in \mathbb{F}^n$, a malicious $\mathcal{P}$ with arbitrary cheating strategies in $\Pi(\mathbb{P}, \mathbb{V})$ convinces $\mathcal{V}$ with probability at most ε.
- **Perfect Zero-Knowledge.** There exists a PPT simulator `Sim` such that, for any arithmetic circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$, any witness $\mathbf{w} \in \mathbb{F}^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$, any r_t, and any $\Delta \in \mathbb{F}$, the output distribution of $\texttt{Sim}(\mathcal{C}, r_t, \Delta)$ is distributed identically to $\mathbf{v}_\Delta$, where $\mathbf{v}(x)$ is the affine line generated in $\Pi(\mathbb{P}, \mathbb{V})$ with $\mathcal{P}$ holding $\mathbf{w}$ and $\mathcal{V}$ sending r_t.

Furthermore, our ILPZK constructions also satisfy the following stronger notions:
Public-coin. An ILPZK proof with algorithms $(\mathbb{P}, \mathbb{V})$ is *public-coin*, if each bit of $\mathcal{V}$'s messages (i.e., r_t, Δ) in $\Pi(\mathbb{P}, \mathbb{V})$ is independently and uniformly random.
ε**-Knowledge Soundness.** An ILPZK proof with algorithms $(\mathbb{P}, \mathbb{V})$ has ε-*knowledge soundness*, if there exists an efficient extractor `Ext` such that, for any arithmetic circuit $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^{n'}$, any r_t selected by $\mathcal{V}$, and any $\mathbf{v}$ generated by (malicious) $\mathcal{P}$ that makes $\mathcal{V}$ accept with $> \varepsilon$ probability, $\texttt{Ext}(\mathcal{C}, r_t, \mathbf{v})$ outputs a valid witness $\mathbf{w}$.

[7] For our constructions in Section 4.1&5.1, we abuse the notation $\mathbf{v}^{(i)}$ for stage i, and denote by $\mathbf{v}^{(i,j)}$ the sub-line generated in round j of stage i.

3.2 Compiling ILPZK to NIZK

Given a t-round *public-coin* ILPZK proof system with algorithms $(\mathbb{P}, \mathbb{V})$, we show how to compile it into a NIZK argument via a two-step approach. As already shown above, from algorithms $(\mathbb{P}, \mathbb{V})$, we can immediately obtain an interactive protocol $\Pi(\mathbb{P}, \mathbb{V})$, assuming the existence of a chosen-input VOLE functionality. In sketch, the first step is to transform $\Pi(\mathbb{P}, \mathbb{V})$ into an interactive protocol $\Pi'(\mathbb{P}, \mathbb{V})$, with a sufficient number of random VOLE correlations generated at the very beginning (say, offline phase). In the second step, we instantiate random VOLE in two ways, yielding two different types of compiling.

In the first step, we rely on the linearly-homomorphism of VOLE. Recall that in each round i of $\Pi(\mathbb{P}, \mathbb{V})$, $\mathcal{V}$ is supposed to learn $\mathbf{v}^{(i)}(\Delta)$. Assume $\mathcal{P}$ holds random $\mathbf{x}^{(i)}, \mathbf{M}^{(i)}$, and $\mathcal{V}$ holds random $\mathbf{K}^{(i)}$ such that $\mathbf{K}^{(i)} = \mathbf{x}^{(i)} \cdot \Delta + \mathbf{M}^{(i)}$. Let $\mathcal{P}$ send $\boldsymbol{\delta}^{(i)} := \mathbf{a}^{(i)} - \mathbf{x}^{(i)}$ and $\boldsymbol{\theta}^{(i)} := \mathbf{b}^{(i)} - \mathbf{M}^{(i)}$ to $\mathcal{V}$, who then can compute

$$\begin{aligned}\mathbf{v}^{(i)}(\Delta) :&= \mathbf{K}^{(i)} + \boldsymbol{\delta}^{(i)} \cdot \Delta + \boldsymbol{\theta}^{(i)} \\ &= (\mathbf{x}^{(i)} \cdot \Delta + \mathbf{M}^{(i)}) + (\mathbf{a}^{(i)} - \mathbf{x}^{(i)}) \cdot \Delta + (\mathbf{b}^{(i)} - \mathbf{M}^{(i)}) \\ &= \mathbf{a}^{(i)} \cdot \Delta + \mathbf{b}^{(i)},\end{aligned}$$

as desired. Since $\mathbf{x}^{(i)}, \mathbf{M}^{(i)}$ are uniformly random, $\mathcal{V}$ learns nothing about $\mathbf{v}^{(i)}(x)$ except for $\mathbf{v}^{(i)}(\Delta)$. By applying the above transformation for every round of $\Pi(\mathbb{P}, \mathbb{V})$, we obtain a t-round public-coin ZK protocol $\Pi'(\mathbb{P}, \mathbb{V})$ in the random VOLE-hybrid model. We briefly state the security that $\Pi'(\mathbb{P}, \mathbb{V})$ could satisfy.

Malicious security in UC-framework. For our ILPZK constructions in this paper, not only the stand-alone security is satisfied, but also the UC-security. Formally speaking, for the ILPZK proof $(\mathbb{P}, \mathbb{V})$ in Section 4.1 or 5.1, the corresponding ZK protocol $\Pi'(\mathbb{P}, \mathbb{V})$ UC-realizes $\mathcal{F}_{\text{ZK}}$ with malicious information-theoretic security in the random VOLE-hybrid model.

Below we introduce the two approaches of the second step.

Instantiating rVOLE with PCG. Boyle et al. [10,12] introduced the cryptographic primitive of pseudorandom correlation generator (PCG), which is an extension of pseudorandom generator (PRG) from generating a batch of randomness to a batch of correlated randomness between some parties. PCG offers a concretely efficient candidate for generating random VOLE correlations in the offline phase. The authors of [11] presented a two-round maliciously secure construction of PCG for VOLE, and showed that one can obtain a designated verifier NIZK from combining it with a non-interactive online phase. Hence, by applying Fiat-Shamir transform to the online phase of $\Pi'(\mathbb{P}, \mathbb{V})$ and instantiating rVOLE with PCG, we can obtain a designated-verifier NIZK argument.

Applying VOLEitH. Given a public-coin rVOLE-based ZK protocol $\Pi'(\mathbb{P}, \mathbb{V})$ defined as above, one can typically obtain a publicly-verifiable NIZK argument through applying the VOLEitH technique proposed by Baum et al. [3]. In a high level, in the VOLEitH framework, random VOLE correlations are instantiated from a (two-round) all-but-one OT protocol, where the prover is the OT sender

and the verifier the OT receiver. Then via the Fiat-Shamir transform, the interactive ZK protocol based on OT is turned into a non-interactive ZK[8].

4 Interactive LPZK for Layered Arithmetic Circuits

In this section, we describe an ILPZK proof system for proving the satisfiability of layered arithmetic circuits, and we show that it achieves all properties as indicated in Theorem 1. In addition, based on this ILPZK, we are able to provide a self-contained VOLE-based ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ in Figure 3. Finally, we prove that, when restricted to layered arithmetic circuits, our protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ UC-realizes $\mathcal{F}_{\mathrm{ZK}}$ in the rVOLE-hybrid model with information-theoretic malicious security.

4.1 Our ILPZK construction

Our ILPZK proof system for layered circuits is inspired by the well-known interactive proof protocol GKR [19], and we employ optimizations from Libra [33] to achieve a linear time prover. In a high level, the core idea of GKR is to reduce a claim about the output layer to a claim about the input layer in an iterative layer-by-layer sense. In [19], the authors discovered a brilliant equation that captures adjacent layers of the circuit, which allows the reduction to be done by employing a sum-check protocol on multi-variate polynomials [24]. For a better readability, let us first explicitly list some notations used in this section.

Notations. In this section, we consider as a layered (log-space uniform) arithmetic circuit over $\mathbb{F}$ of depth d, size S, and fan-in two. Each layer of $\mathcal{C}$ is labeled by a number from 0 to d, with 0 being the output layer and d being the input layer. More precisely, in a layered circuit $\mathcal{C}$, layer $i \in [0, d)$ consists of add/mult gates, which take input from outputs of layer $i+1$, and layer d consists of input gates. W.l.o.g., we always assume each layer i of $\mathcal{C}$ contains in total $s_i = 2^{k_i}$ gates (thus $S = \sum_{i=0}^{d-1} s_i$), and we label them in a pre-defined order. In addition, given the pre-defined order, we denote values on the output wires of gates in each layer i by $\mathbf{W}_i \in \mathbb{F}^{s_i}$. We also define two functions for each layer $i \in [0, d)$, $\mathrm{add}_i, \mathrm{mult}_i : \{0,1\}^{k_i+2k_{i+1}} \to \{0,1\}$, referred as "wiring predicates". Each add_i (mult_i) takes one gate label $z \in \{0,1\}^{k_i}$ in layer i and two gate labels $x, y \in \{0,1\}^{k_{i+1}}$ in layer $i+1$, and outputs 1 if and only if gate z in layer i is an addition (multiplication) gate that takes the output of gate x, y in layer $i+1$ as input. We view each $\mathbf{W}_i \in \mathbb{F}^{s_i}$ as a function $W_i : \{0,1\}^{k_i} \to \mathbb{F}$, and denote the multi-linear extension of $W_i, \mathrm{mult}_i, \mathrm{add}_i$ by $\widetilde{W}_i, \widetilde{\mathrm{mult}}_i, \widetilde{\mathrm{add}}_i$, respectively.

With $W_i, \mathrm{mult}_i, \mathrm{add}_i$ defined as above, it holds for all $i \in [0, d)$ that

$$W_i(z) = \sum_{x,y \in \{0,1\}^{k_{i+1}}} \mathrm{mult}_i(z,x,y) W_{i+1}(x) W_{i+1}(y) + \mathrm{add}_i(z,x,y)\big(W_{i+1}(x) + W_{i+1}(y)\big),$$

[8] We remark that for statements defined over large fields, we need to apply the so-called *subspace VOLE* technique of [3] to significantly reduce the prover computation on generating random VOLE correlations. We omit the details as they are not the focus of this work, and refer to [26,3].

where $z \in \{0,1\}^{k_i}$. This immediately implies the following equation

$$\widetilde{W}_i(z) = \sum_{x,y \in \{0,1\}^{k_{i+1}}} \widetilde{\mathrm{mult}}_i(z,x,y)\widetilde{W}_{i+1}(x)\widetilde{W}_{i+1}(y) + \widetilde{\mathrm{add}}_i(z,x,y)\big(\widetilde{W}_{i+1}(x) + \widetilde{W}_{i+1}(y)\big), \tag{1}$$

holds for all $z \in \mathbb{F}^{k_i}$.

Our ILPZK is also within the commit-and-prove paradigm. In the "commit" phase, the prover commits to the values that capture the full evaluation of $\mathcal{C}(\mathbf{w})$ (also depend on messages received from the verifier), while in the "prove" phase, the prover "opens" those commitments in zero-knowledge, thus the verifier can check whether the prover holds a witness $\mathbf{w}$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

In a high level, our ILPZK proof for layered circuits can be divided into d sequential stages, each consisting of a "sub-commit" phase and a "sub-prove" phase (jumping ahead, each stage contains several rounds in our constructions). Intuitively, in each stage $i \in [0,d)$, the prover $\mathcal{P}$ proves a sub-statement (indexed by i) to the verifier $\mathcal{V}$ that he knows some $\widetilde{W}_{i+1}$ such that Eq.(1) holds for $\widetilde{W}_i$ evaluated at a random point $\boldsymbol{r}_i \in \mathbb{F}^{k_i}$ chosen by $\mathcal{V}$. However, the sub-statement i is never completely proved, unless $\mathcal{P}$ proves the sub-statement $i+1$. Until they reach stage $d-1$, $\mathcal{P}$ completely prove to $\mathcal{V}$ that he knows the witness $\widetilde{W}_d$ such that Eq.(1) holds for $\widetilde{W}_{d-1}$ evaluated at a random point $\boldsymbol{r}_{d-1} \in \mathbb{F}^{k_{d-1}}$ chosen by $\mathcal{V}$. Eventually, these d sub-proofs together convince $\mathcal{V}$ that $\mathcal{P}$ knows a witness $\mathbf{w}$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

Since the underlying commitment scheme is statistical binding, the sub-prove phase of each stage $i \in [0,d)$ can be deferred to stage $d-1$ (i.e., when all sub-commit phases are completed). Hence, let us first describe how $\mathcal{P}$ and $\mathcal{V}$ proceed in the sub-commit phase of each stage i, and explain the arithmetic constraints that values underneath the commitments should satisfy. We remark that in fact, details of the first and the last stages are slightly different from the rest of the stages, which we will explain later. Essentially, $\mathcal{P}$ and $\mathcal{V}$ perform a two-phase (linear time) sum-check protocol on Eq.(1) underneath the commitment in each stage i.

In each stage $i \in [0,d)$, suppose that $\mathcal{P}$ and $\mathcal{V}$ start with an agreement on $\boldsymbol{r}_i$ and a commitment $v_i := \widetilde{W}_i(\boldsymbol{r}_i) \cdot \Delta + b_i$ (with $\mathcal{P}$ holds $\widetilde{W}_i(\boldsymbol{r}_i), b_i$ and $\mathcal{V}$ holds v_i, Δ), denoted by $[\widetilde{W}_i(\boldsymbol{r}_i)]$, where $\boldsymbol{r}_i \in \mathbb{F}^{k_i}$ is a random point selected by $\mathcal{V}$ (jumping ahead, $\boldsymbol{r}_i$ is determined over several rounds of interactions during the previous stage). At the end of stage i, they will agree on a point $\boldsymbol{r}_{i+1} \in \mathbb{F}^{k_{i+1}}$ and obtain a commitment $v_{i+1} := \widetilde{W}_{i+1}(\boldsymbol{r}_{i+1}) \cdot \Delta + b_{i+1}$, denoted by $[\widetilde{W}_{i+1}(\boldsymbol{r}_{i+1})]$, so that they can move to the next stage.

According to Eq. (1), $\mathcal{P}$ can define a $2k_{i+1}$-variate polynomial

$$f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X},\mathbf{Y}) := \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i,\mathbf{X},\mathbf{Y})\widetilde{W}_{i+1}(\mathbf{X})\widetilde{W}_{i+1}(\mathbf{Y}) + \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i,\mathbf{X},\mathbf{Y})\big(\widetilde{W}_{i+1}(\mathbf{X}) + \widetilde{W}_{i+1}(\mathbf{Y})\big). \tag{2}$$

Essentially, $\mathcal{P}$ wants to convince $\mathcal{V}$ that $\sum_{x,y \in \{0,1\}^{k_{i+1}}} f^{(i)}_{\boldsymbol{r}_i}(x,y) = \widetilde{W}_i(\boldsymbol{r}_i)$. We let $\mathcal{P}$ and $\mathcal{V}$ interact $2k_{i+1}$ rounds to capture the summation of $f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X},\mathbf{Y})$ evaluated on the binary cube. Intuitively in each round, they sum-up one variable of

$f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X}, \mathbf{Y})$. For the first k_{i+1} rounds, $\mathcal{P}$ defines two univariate polynomials

$$A_{\boldsymbol{r}_i}(\mathbf{X}) := \sum_{y \in \{0,1\}^{k_{i+1}}} \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \mathbf{X}, y) \cdot \widetilde{W}_{i+1}(y) + \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \mathbf{X}, y),$$

and

$$B_{\boldsymbol{r}_i}(\mathbf{X}) := \sum_{y \in \{0,1\}^{k_{i+1}}} \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \mathbf{X}, y) \cdot \widetilde{W}_{i+1}(y).$$

This immediately implies that

$$\sum_{x,y \in \{0,1\}^{k_{i+1}}} f^{(i)}_{\boldsymbol{r}_i}(x, y) = \sum_{x \in \{0,1\}^{k_{i+1}}} A_{\boldsymbol{r}_i}(x) \cdot \widetilde{W}_{i+1}(x) + B_{\boldsymbol{r}_i}(x).$$

In round j[9], where $j = 1, \ldots, k_{i+1}$, $\mathcal{P}$ can compute a univariate polynomial $g^{(i,j)}(X_j)$ from $A_{\boldsymbol{r}_i}(\mathbf{X}), \widetilde{W}_{i+1}(\mathbf{X}), B_{\boldsymbol{r}_i}(\mathbf{X})$ as follows:

$$\begin{aligned} g^{(i,j)}(X_j) &:= \sum_{x_{j+1},\ldots,x_{k_{i+1}} \in \{0,1\}} (A_{\boldsymbol{r}_i} \cdot \widetilde{W}_{i+1} + B_{\boldsymbol{r}_i})(\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_{j-1}, X_j, x_{j+1}, \ldots, x_{k_{i+1}}) \\ &= \sum_{x_{j+1},\ldots,x_{k_{i+1}} \in \{0,1\}} \sum_{y \in \{0,1\}^{k_{i+1}}} f^{(i)}_{\boldsymbol{r}_i}(\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_{j-1}, X_j, x_{j+1}, \ldots, x_{k_{i+1}}, y), \end{aligned} \tag{3}$$

where $\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_{j-1} \in \mathbb{F}$ are sent by $\mathcal{V}$ in the previous $j-1$ rounds. Since $A_{\boldsymbol{r}_i}(\mathbf{X})$ and $\widetilde{W}_{i+1}(\mathbf{X})$ are multi-linear polynomials, $g^{(i,j)}(X_j)$ has degree 2. Then $\mathcal{P}$ computes three coefficients of $g^{(i,j)}(X_j)$, denoted by $\mathbf{g}^{(i,j)} := (g^{(i,j)}_0, g^{(i,j)}_1, g^{(i,j)}_2)$, and samples $\mathbf{b}^{(i,j)} \xleftarrow{\$} \mathbb{F}^3$. The interaction of round j proceeds as follows:

- $\mathcal{P}$ sends $(\mathbf{g}^{(i,j)}, \mathbf{b}^{(i,j)})$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{VOLE}}$, which then returns $\mathbf{v}^{(i,j)}_{\Delta} := \mathbf{g}^{(i,j)} \cdot \Delta + \mathbf{b}^{(i,j)}$ to $\mathcal{V}$.
- Upon receiving $\mathbf{v}^{(i,j)}_{\Delta}$, $\mathcal{V}$ sends $\bar{x}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

Essentially, $\mathcal{P}$ commits to a polynomial $g^{(i,j)}(X_j)$, denoted by $[\![g^{(i,j)}(\cdot)]\!]$, from which they can obtain $[g^{(i,j)}(\alpha)]$ for any given $\alpha \in \mathbb{F}$. This is crucial for the sub-prove phase, and below we show arithmetic constraints on these $[\![g^{(i,j)}(\cdot)]\!]$. Recall that in the first round, $\mathcal{P}$ builds an affine line $\mathbf{v}^{(i,1)}(x) := \mathbf{g}^{(i,1)} \cdot x + \mathbf{b}^{(i,1)}$, where $\mathbf{g}^{(i,1)}$ contains coefficients of $g^{(i,1)}(X_1)$. By definitions of $g^{(i,1)}, \widetilde{W}_i$, we have that $\widetilde{W}_i(\boldsymbol{r}_i) = g^{(i,1)}(0) + g^{(i,1)}(1)$, which gives a linear constraint: given $[\widetilde{W}_i(\boldsymbol{r}_i)], [\![g^{(i,1)}(\cdot)]\!]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$\widetilde{W}_i(\boldsymbol{r}_i) = 2g^{(i,1)}_0 + g^{(i,1)}_1 + g^{(i,1)}_2.$$

In round $j \in [2, k_{i+1}]$, $\mathcal{P}$ builds an affine line $\mathbf{v}^{(i,j)}(x) := \mathbf{g}^{(i,j)} \cdot x + \mathbf{b}^{(i,j)}$. By definitions of $g^{(i,j-1)}(X_{j-1}), g^{(i,j)}(X_j)$ (Eq.(3)), we have that $g^{(i,j-1)}(\bar{x}^{(i)}_{j-1}) =$

[9] Here the interaction where $\mathcal{P}$ first sends a line, and then $\mathcal{V}$ replies a challenge is viewed as one round, which is consistent with our ILPZK definition (Def. 2).

$g^{(i,j)}(0) + g^{(i,j)}(1)$, which gives a linear constraint: given $[\![g^{(i,j-1)}(\cdot)]\!]$, $[\![g^{(i,j)}(\cdot)]\!]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$g_0^{(i,j-1)} + g_1^{(i,j-1)} \cdot \bar{x}_{j-1}^{(i)} + g_2^{(i,j-1)} \cdot (\bar{x}_{j-1}^{(i)})^2 = 2g_0^{(i,j)} + g_1^{(i,j)} + g_2^{(i,j)}.$$

Now we move on to round $k_{i+1}+j$, where $j = 1, \ldots, k_{i+1}$, $\mathcal{P}$ instead computes a univariate polynomial $h^{(i,j)}(Y_j)$ with degree-2 simply from $f_{\boldsymbol{r}_i}^{(i)}(\mathbf{X}, \mathbf{Y})$:

$$h^{(i,j)}(Y_j) := \sum_{y_{j+1},\ldots,y_{k_{i+1}} \in \{0,1\}} f_{\boldsymbol{r}_i}^{(i)}(\bar{\boldsymbol{x}}^{(i)}, \bar{y}_1^{(i)}, \ldots, \bar{y}_{j-1}^{(i)}, Y_j, y_{j+1}, \ldots, y_{k_{i+1}}), \quad (4)$$

where $\bar{\boldsymbol{x}}^{(i)}, \bar{y}_1^{(i)}, \ldots, \bar{y}_{j-1}^{(i)}$ are sent by $\mathcal{V}$ in the previous $(k_{i+1} + j - 1)$ rounds. Similarly, $\mathcal{P}$ computes $\mathbf{h}^{(i,j)}$ and samples $\mathbf{b}^{(i,k_{i+1}+j)} \xleftarrow{\$} \mathbb{F}^3$. The interaction of round $(k_{i+1} + j)$ proceeds as follows:

- $\mathcal{P}$ sends $(\mathbf{h}^{(i,j)}, \mathbf{b}^{(i,k_{i+1}+j)})$ to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$, which then returns to $\mathcal{V}$ a $\mathbf{v}_{\Delta}^{(i,k_{i+1}+j)} := \mathbf{h}^{(i,j)} \cdot \Delta + \mathbf{b}^{(i,k_{i+1}+j)}$.
- Upon receiving $\mathbf{v}_{\Delta}^{(i,k_{i+1}+j)}$, $\mathcal{V}$ sends $\bar{y}_j^{(i)} \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

Intuitively, $\mathcal{P}$ commits to polynomials $h^{(i,j)}(Y_j)$ for $j \in [1, k_{i+1}]$, and below we show arithmetic constraints on these $[\![h^{(i,j)}(\cdot)]\!]$. Very similarly, in the round $(k_{i+1} + 1)$, we have the following linear constraint: given $[\![g^{(i,k_{i+1})}(\cdot)]\!]$, $[\![h^{(i,1)}(\cdot)]\!]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$g_0^{(i,k_{i+1})} + g_1^{(i,k_{i+1})} \cdot \bar{x}_{k_{i+1}}^{(i)} + g_2^{(i,k_{i+1})} \cdot (\bar{x}_{k_{i+1}}^{(i)})^2 = 2h_0^{(i,1)} + h_1^{(i,1)} + h_2^{(i,1)}.$$

And in the following round $(k_{i+1} + j)$, where $j \in [2, k_{i+1}]$, we have the following linear constraint: given $[\![h^{(i,j-1)}(\cdot)]\!]$, $[\![h^{(i,j)}(\cdot)]\!]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$h_0^{(i,j-1)} + h_1^{(i,j-1)} \cdot \bar{y}_{j-1}^{(i)} + h_2^{(i,j-1)} \cdot (\bar{y}_{j-1}^{(i)})^2 = 2h_0^{(i,j)} + h_1^{(i,j)} + h_2^{(i,j)}.$$

Note that after round $2k_{i+1}$, the prover $\mathcal{P}$ has committed to a polynomial $h^{(i,k_{i+1})}(Y_{k_{i+1}})$, which leads to a commitment $[h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)})]$. By definition, we have $h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)}) = f_{\boldsymbol{r}_i}^{(i)}(\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$. Suppose $\mathcal{P}$ has committed to $\widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)})$, $\widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})$. According to Eq.(2), we have the following degree-2 constraint: given $[h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)})]$, $[\widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)})]$, $[\widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$\begin{aligned} h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)}) = {} & \widetilde{\text{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \cdot \widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)}) \cdot \widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)}) \\ & + \widetilde{\text{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \cdot \big(\widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)}) + \widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})\big). \end{aligned}$$

In order to move to the next stage, we could let $\mathcal{P}$ and $\mathcal{V}$ perform stage $i+1$ twice on respective inputs $[\widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)})]$, $[\widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})]$. However, this naive approach will incur an exponential blow-up in the circuit depth d.

To avoid this issue, we adapt the technique proposed in GKR [19], which allows for combining the above two sub-statements to one sub-statement $i + 1$.

The strategy requires one more round of interaction, and employs an observation that $\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)} \in \mathbb{F}^{k_{i+1}}$ determine an affine line $L^{(i)}$ such that $L^{(i)}(0) = \bar{\boldsymbol{x}}^{(i)}$ and $L^{(i)}(1) = \bar{\boldsymbol{y}}^{(i)}$. Then $\mathcal{P}$ can obtain a univariate polynomial $q^{(i)}(\cdot)$ with degree-k_{i+1} by restricting $\widetilde{W}_{i+1}(\cdot)$ to $L^{(i)}$, which satisfies that $q^{(i)}(0) = \widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)})$ and $q^{(i)}(1) = \widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})$. Therefore, by $\mathcal{P}$ committing to $q^{(i)}(X)$, the two commitments $[\widetilde{W}_{i+1}(\bar{\boldsymbol{x}}^{(i)})]$, $[\widetilde{W}_{i+1}(\bar{\boldsymbol{y}}^{(i)})]$ can be replaced by $[q^{(i)}(0)]$, $[q^{(i)}(1)]$. We let $\mathcal{P}$ compute $\mathbf{q}^{(i)}$ and sample $\mathbf{b}^{(i,2k_{i+1}+1)} \xleftarrow{\$} \mathbb{F}^{k_{i+1}+1}$. Now the additional round $2k_{i+1}+1$ proceeds as follows:

- $\mathcal{P}$ sends $(\mathbf{q}^{(i)}, \mathbf{b}^{(i,2k_{i+1}+1)})$ to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$, which then returns to $\mathcal{V}$ a $\mathbf{v}_{\Delta}^{(i,2k_{i+1}+1)} := \mathbf{q}^{(i)} \cdot \Delta + \mathbf{b}^{(i,2k_{i+1}+1)}$.
- Upon receiving $\mathbf{v}_{\Delta}^{(i,2k_{i+1}+1)}$, $\mathcal{V}$ sends $r^{(i)} \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

Now the corresponding degree-2 constraint is as follows: given $[\![h^{(i,k_{i+1})}(\cdot)]\!]$, $[\![q^{(i)}(\cdot)]\!]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$\sum_{l=0}^{2} h_l^{(i,k_{i+1})} (\bar{y}_{k_{i+1}}^{(i)})^l = \widetilde{\text{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) q_0^{(i)} (\sum_{l=0}^{k_{i+1}} q_l^{(i)}) + \widetilde{\text{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \big(q_0^{(i)} + (\sum_{l=0}^{k_{i+1}} q_l^{(i)})\big)$$

As for moving to the next stage, they can set $\boldsymbol{r}_{i+1} := L^{(i)}(r^{(i)})$, with $\mathcal{P}$, $\mathcal{V}$ locally computing

$$b_{i+1} := \sum_{j=0}^{k_{i+1}} b_j^{(i,2k_{i+1}+1)} \cdot (r^{(i)})^j, \;\; v_{i+1} := \sum_{j=0}^{k_{i+1}} v_{\Delta,j}^{(i,2k_{i+1}+1)} \cdot (r^{(i)})^j,$$

respectively. Note that it is supposed to hold that $v_{i+1} = \widetilde{W}_{i+1}(\boldsymbol{r}_{i+1}) \cdot \Delta + b_{i+1}$, (i.e., they can obtain $[\widetilde{W}_{i+1}(\boldsymbol{r}_{i+1})]$ in this way). For simplicity, we denote $(\mathbf{v}^{(i,1)}(x), \ldots, \mathbf{v}^{(i,2k_{i+1}+1)}(x))$ by $\mathbf{v}^{(i)}(x)$.

In the first stage, $\mathcal{P}$ and $\mathcal{V}$ need to run a setup at first. According to the definition of ILPZK, $\mathcal{V}$ picks a random $\Delta \in \mathbb{F}$ and sends it to $\mathcal{F}_{\text{VOLE}}$. Jumping ahead, $\mathcal{P}$ will be required to generate a line that captures the witness $\mathbf{w}$ before he received challenges in the last stage. As the line is only dependent on the witness, we let $\mathcal{P}$ do this at the very beginning of stage 0. Thus, the one round of interaction for setup proceeds as follows:

- $\mathcal{P}$ samples $\mathbf{b}^{(0,0)} \xleftarrow{\$} \mathbb{F}^{s_d}$, and sends $(\mathbf{w}, \mathbf{b}^{(0,0)})$ to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$. Then $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$ returns $\mathbf{v}_{\Delta}^{(0,0)} := \mathbf{w} \cdot \Delta + \mathbf{b}^{(0,0)}$ to $\mathcal{V}$.
- Upon receiving $\mathbf{v}_{\Delta}^{(0,0)}$, $\mathcal{V}$ sends $\boldsymbol{r}_0 \xleftarrow{\$} \mathbb{F}^{k_0}$ to $\mathcal{P}$.

Note that as in this paper we focus on the circuit satisfiability problem, the output values of layer 0 should be all 1's (i.e., $\mathbf{W}_0 = \mathbf{1} \in \mathbb{F}^{s_0}$) as long as $\mathcal{P}$ inputs a witness $\mathbf{w}$. Hence, we let $\mathcal{V}$ locally set $v_0 = \widetilde{W}_0(\boldsymbol{r}_0) \cdot \Delta$, and $\mathcal{P}$ locally set $b_0 = 0$. This completes the setup, and they can move on. For simplicity, we let $\mathbf{v}^{(0)}(x)$ also include $\mathbf{v}^{(0,0)}(x)$.

In the last stage, to complete the whole proof, $\mathcal{P}$ remains to convince $\mathcal{V}$ that all previous affine lines $\mathbf{v}^{(0)}(x), \ldots, \mathbf{v}^{(d-1)}(x)$ are honestly generated (i.e., run the deferred sub-prove phases.). In addition to the arithmetic constraints we have specified above, there is an arithmetic constraint on $[\widetilde{\mathbf{W}}_d(\boldsymbol{r}_d)]$ (obtained in stage $d-1$) and $[\mathbf{W}_d]$ (obtained in stage 0), where $\mathcal{P}$ needs to convince $\mathcal{V}$ that

$$\widetilde{W}_d(\boldsymbol{r}_d) = \sum_{\omega \in \{0,1\}^{k_d}} W_d(\omega) \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d),$$

which is induced by the Lagrange interpolation of Def 1.

So far we have shown the arithmetic constraints that $\mathbf{v}^{(0)}(x), \ldots, \mathbf{v}^{(d-1)}(x)$ need to satisfy are actually degree-2 constraints. Therefore, we can introduce one more round of interaction to efficiently check these constraints in a batch, as indicated in the end of Section 2.4. For simplicity, we let $\mathbf{v}^{(d-1)}(x)$ include the extra two entries for batch checking linear and multiplication constraints.

4.2 Complexity

Here we analyze the complexity of our construction in Section 4.1.

Round complexity. Our construction can be divided into d stages, with each stage $i \in [1, d-2]$ having $2k_{i+1}+1$ rounds. In particular, stage 0 has $2k_1+2$ rounds, and stage $d-1$ has $2k_d+2$ rounds. Thus, there are $2+\sum_{i=1}^{d}(k_i+1) = \mathcal{O}(d \log S)$ rounds in total.

Proof size. The proof size is the summation of length of $\mathbf{v}^{(0)}(x), \ldots, \mathbf{v}^{(d-1)}(x)$. For $i \in [1, d-2]$, each $\mathbf{v}^{(i)}$ has length $6k_{i+1} + k_{i+1} + 1 = 7k_{i+1} + 1$, while $\mathbf{v}^{(0)}(x)$ has length $s_d + 7k_1 + 1$, and $\mathbf{v}^{(d-1)}(x)$ has length $7k_d + 1 + 2$. Thus, the total proof size is $\mathcal{O}\Big(s_d + 2 + \sum_{i=1}^{d}(7k_i+1)\Big) = \mathcal{O}(n + d \log S)$.

Prover time. Applying "sum-check" dominates the prover time. By Lemma 2, for each stage i, performing a two-phase sum-check for generating sub-statements costs prover time $2\sum_{j=0}^{k_{i+1}} \mathcal{O}\big(2^{k_{i+1}-j}\big) = \mathcal{O}\big(2^{k_{i+1}}\big) = \mathcal{O}(s_{i+1})$, and according to [33], computing the univariate polynomial $q^{(i)}(\cdot)$ of degree-k_{i+1} can be done in $\mathcal{O}\big(2^{k_{i+1}}\big) = \mathcal{O}(s_{i+1})$ time. Thus, it takes the prover overall $\sum_{i=0}^{d-1}(\mathcal{O}(s_{i+1}) + \mathcal{O}(s_{i+1})) = \mathcal{O}(S)$ time.

Verifier time. The verifier $\mathcal{V}$ needs to at least read the entire proof, which takes time $\mathcal{O}(n + d \log S)$. In addition, $\mathcal{V}$ also needs to evaluate multi-linear polynomials at some specific points, including computing $[\widetilde{W}_0(\boldsymbol{r}_0)]$, $[\widetilde{W}_d(\boldsymbol{r}_d)]$ and $\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$, $\widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$ for each $i \in [0, d)$. By Lemma 1, the former two computations cost time in total $\mathcal{O}(n + s_0)$. Evaluating $\widetilde{\mathrm{mult}}_i$ and $\widetilde{\mathrm{add}}_i$ can be done in time $\mathcal{O}(\log S)$ for several types of circuits [14,28], as they are usually very sparse. Also, [19] proposed a method for log-space uniform circuit, which takes verifier time $\mathcal{O}(d \log S)$ by outsourcing the computation to the prover. In summary, for layered log-space uniform circuits, $\mathcal{V}$ runs in time $\mathcal{O}(n + s_0 + d \log S)$.

4.3 Security Proof in UC-framework

We present in Figure 3 a self-contained zero-knowledge protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ in the random VOLE-hybrid model, which is based on our ILPZK proof in Section 4.1.

In the offline phase, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ invoke the random VOLE functionality $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$, and they obtain a certain number of random VOLE correlations (i.e., VOLE commitments of random values). Due to the linearity of VOLE correlations, $\mathcal{P}$ can commit to a value w by sending $\delta := w - \nu$ to $\mathcal{V}$ and computing $[w] := [\nu] + \delta$, where $[\nu]$ is a random VOLE correlation.

In the online phase, $\mathcal{P}$ and $\mathcal{V}$ follow the instructions of our ILPZK construction except that $\mathcal{P}$ commits to values through random VOLE instead of VOLE. For a cleaner presentation, we design two tailored procedures for batch-checking linear and multiplicative constraints, with details in Figure 2. In the case of checking linear constraints, it suffices to allow $\mathcal{V}$ to check the value underneath a certain commitment is zero. More specifically, given a VOLE-based commitment $[x]$ with $K_x = x \cdot \Delta + M_x$, $\mathcal{V}$ can check $[0] \stackrel{?}{=} [x]$ by $\mathcal{P}$ revealing M_x and checking $K_x \stackrel{?}{=} M_x$. Hence, applying this procedure would not increase the number of random VOLE correlations. While for the case of checking multiplicative constraints, they need to consume one random VOLE correlation. More specifically, recall the multiplicative constraint check in Section 2.4, $\mathcal{V}$ needs to obtain $v_4(\Delta)$ so that she can complete the verification. But directly revealing the two coefficients of $v_4(x)$ certainly leaks information about $\mathcal{P}$'s inputs. To prevent this leakage, one can mask it by an additional entry of random VOLE. Therefore, by using the random linear combination technique, performing the two checking procedures only consumes one additional entry of random VOLE.

The following theorem asserts the security of our protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ with its proof deferred to Appendix B.1. Moreover, Theorem 3 implies Theorem 1.

Theorem 3. *Our ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ UC-realizes the ZK functionality $\mathcal{F}_{\mathrm{ZK}}$ in the $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$-hybrid model with information-theoretic malicious security. In particular, the environment $\mathcal{Z}$'s advantage is $\mathcal{O}\Big(\frac{d \log S}{|\mathbb{F}|}\Big)$.*

Extending to any field. Recall that random subfield VOLE allows to commit elements of a small field $\mathbb{F}_p$ over a large enough extension field $\mathbb{F}_{p^r}$. Therefore, to prove the satisfiability of a circuit $\mathcal{C}$ over $\mathbb{F}_p$, one can substitute VOLE with subfield VOLE for "gate-by-gate" VOLE-based ZK protocols as all wire values are over $\mathbb{F}_p$. However, for our "layer-by-layer" protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$, the multi-linear extensions of layers must be evaluated at $\mathbb{F}_{p^r}$ points for security guarantee (hence are over $\mathbb{F}_{p^r}$), while the witness $\mathbf{w}$ is over $\mathbb{F}_p$. Therefore, we intuitively hope for a mixture of random VOLE and random subfield VOLE, where the verifier holds the same random Δ. We formalize the required functionality as $\mathcal{F}^{\mathbb{F}_p,\mathbb{F}_{p^r}}_{\mathrm{sVOLE}}$ in Figure 11. On top of $\mathcal{F}^{\mathbb{F}_p,\mathbb{F}_{p^r}}_{\mathrm{sVOLE}}$, we can easily extend our ZK protocols to any field and for completeness we present the adaption of the above protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$ in Figure 12 in Appendix D. It remains to show an efficient construction of $\mathcal{F}^{\mathbb{F}_p,\mathbb{F}_{p^r}}_{\mathrm{sVOLE}}$. In fact, we observe that by fixing a basis of $\mathbb{F}_{p^r}$ over $\mathbb{F}_p$, denoted by $\lambda_1, \ldots, \lambda_r$, standard VOLE correlations can be locally computed from subfield VOLE correlations. For

Procedure `Batch-Lin`: on input $\{[m_i], [\![g^{(i,j)}]\!], [\![h^{(i,j)}]\!]\}_{i\in[0,d),j\in[1,k_{i+1}]}$.

1. For $i \in [0, d-1]$, $j \in [1, k_{i+1}]$, $\mathcal{P}$ and $\mathcal{V}$ compute $([g^{(i,j)}(0)]$, $[g^{(i,j)}(1)]$, $[g^{(i,j)}(\bar{x}_j^{(i)})])$ and $([h^{(i,j)}(0)]$, $[h^{(i,j)}(1)]$, $[h^{(i,j)}(\bar{y}_j^{(i)})])$.
2. The verifier $\mathcal{V}$ wants checks that $[0] \stackrel{?}{=} [m_i] - [g^{(i,1)}(0)] - [g^{(i,1)}(1)]$, $[0] \stackrel{?}{=} [g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})] - [h^{(i,1)}(0)] - [h^{(i,1)}(1)]$ and $[0] \stackrel{?}{=} [g^{(i,j)}(\bar{x}_j^{(i)})] - [g^{(i,j+1)}(0)] - [g^{(i,j+1)}(1)]$, $[0] \stackrel{?}{=} [h^{(i,j)}(\bar{y}_j^{(i)})] - [h^{(i,j+1)}(0)] - [h^{(i,j+1)}(1)]$ for $j \in [1, k_{i+1})$. For simplicity, we naturally view these $N := 2\sum_{i=1}^{d} k_i$ constraints as $([x_i], [y_i], [z_i])$ such that $z_i = x_i + y_i$ should hold for all $i \in [N]$.
3. $\mathcal{V}$ samples $\boldsymbol{\lambda} \xleftarrow{\$} \mathbb{F}^N$, and sends it to $\mathcal{P}$.
4. $\mathcal{P}$ computes $M_{\text{lin}} := \sum_{i=1}^{N} \lambda_i (M_{z_i} - M_{x_i} - M_{y_i})$, and sends it to $\mathcal{V}$.
5. $\mathcal{V}$ computes $K_{\text{lin}} := \sum_{i=1}^{N} \lambda_i (K_{z_i} - K_{x_i} - K_{y_i})$, and checks that $K_{\text{lin}} \stackrel{?}{=} M_{\text{lin}}$.

Procedure `Batch-Mult`: on input $\{[\![q^{(i)}(\cdot)]\!], [\![h^{(i,k_{i+1})}(\cdot)]\!]\}_{i\in[0,d)}$.

1. For $i \in [0, d-1]$, $\mathcal{P}$ and $\mathcal{V}$ compute $[x_i] := [q^{(i)}(0)]$, $[y_i] := [q^{(i)}(1)]$, $[z_i] := [h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)})]$, and $a_i := \widetilde{\text{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$, $b_i := \widetilde{\text{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$.
2. $\mathcal{V}$ wants to check that $z_i \stackrel{?}{=} a_i x_i y_i + b_i(x_i + y_i)$ for all i.
3. $\mathcal{V}$ samples $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}^d$ and sends it to $\mathcal{P}$.
4. They consume a random VOLE correlation $[\pi]$ in the sense that $\mathcal{P}$ computes $M_{\text{mult}} := M_\pi + \sum_{i=0}^{d-1} \alpha_i (a_i M_{x_i} M_{y_i})$, and

$$x_{\text{mult}} := \pi + \sum_{i=0}^{d-1} \alpha_i (a_i (y_i M_{x_i} + x_i M_{y_i}) + b_i (M_{x_i} + M_{y_i}) - M_{z_i}),$$

   while $\mathcal{V}$ computes

$$K_{\text{mult}} := K_\pi + \sum_{i=0}^{d-1} \alpha_i (a_i K_{x_i} K_{y_i} + (b_i (K_{x_i} + K_{y_i}) - K_{z_i}) \cdot \Delta).$$

5. $\mathcal{P}$ sends $(x_{\text{mult}}, M_{\text{mult}})$ to $\mathcal{V}$, who then checks that $K_{\text{mult}} \stackrel{?}{=} M_{\text{mult}} + x_{\text{mult}} \cdot \Delta$.

Fig. 2: Procedures for batch-checking linear and multiplicative constraints.

Protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$

Notations follow Section 4.1. The prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that he holds a witness $\mathbf{w} \in \mathbb{F}^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

Offline phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ send $(\texttt{Init})$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$, and $\mathcal{V}$ receives $\Delta \in \mathbb{F}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{Extend}, n + \sum_{i=1}^{d}(7k_i + 1) + 1)$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$, which returns commitments on random values, denoted by $[\boldsymbol{\nu}], [\boldsymbol{\mu}], [\pi]$, where $\boldsymbol{\nu} \in \mathbb{F}^n$, $\pi \in \mathbb{F}$. For simplicity, we view $\boldsymbol{\mu}$ as $\{\mu_{i,j}\}_{i\in[0,d),j\in[7k_{i+1}+1]}$.

Online phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ obtain $[\mathbf{w}]$ (by $\mathcal{P}$ sending $\boldsymbol{\delta} := \mathbf{w} - \boldsymbol{\nu}$ to $\mathcal{V}$).
2. For each layer i, $\mathcal{P}$ computes $\mathbf{W}_i$ and stores them. $\mathcal{V}$ sends a random $\boldsymbol{r}_0 \in \mathbb{F}^{k_0}$ to $\mathcal{P}$, then they locally compute $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$ (Note that $\mathbf{W}_0 = \mathbf{1} \in \mathbb{F}^{s_0}$).
3. For $i = 0, 1, \ldots, d-1$,
   (a) The prover $\mathcal{P}$ defines the $2k_{i+1}$-variate polynomial $f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X}, \mathbf{Y}) := \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y})\widetilde{W}_{i+1}(\mathbf{X})\widetilde{W}_{i+1}(\mathbf{Y}) + \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y})(\widetilde{W}_{i+1}(\mathbf{X}) + \widetilde{W}_{i+1}(\mathbf{Y}))$.
   (b) For $j = 1, \ldots, k_{i+1}$,
      i. $\mathcal{P}$ computes a univariate polynomial $g^{(i,j)}(X_j)$ of degree-2, writing as $g^{(i,j)}_0 + g^{(i,j)}_1 \cdot X_j + g^{(i,j)}_2 \cdot X_j^2$. $\mathcal{P}$ sends $g^{(i,j)}_0 - \mu_{i,3j-2}$, $g^{(i,j)}_1 - \mu_{i,3j-1}$, $g^{(i,j)}_2 - \mu_{i,3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([g^{(i,j)}_0], [g^{(i,j)}_1], [g^{(i,j)}_2])$, denoted by $[\![g^{(i,j)}(\cdot)]\!]$.
      ii. $\mathcal{V}$ samples $\bar{x}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
   (c) For $j = 1, \ldots, k_{i+1}$,
      i. $\mathcal{P}$ computes a single variable polynomial $h^{(i,j)}(Y_j)$ of degree 2, writing as $h^{(i,j)}_0 + h^{(i,j)}_1 \cdot Y_j + h^{(i,j)}_2 \cdot Y_j^2$. $\mathcal{P}$ sends $h^{(i,j)}_0 - \mu_{i,3k_{i+1}+3j-2}$, $h^{(i,j)}_1 - \mu_{i,3k_{i+1}+3j-1}$, $h^{(i,j)}_2 - \mu_{i,3k_{i+1}+3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([h^{(i,j)}_0], [h^{(i,j)}_1], [h^{(i,j)}_2])$, denoted by $[\![h^{(i,j)}(\cdot)]\!]$.
      ii. $\mathcal{V}$ samples $\bar{y}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
   (d) Let $L^{(i)}$ be the unique line satisfying $L^{(i)}(0) = \bar{\boldsymbol{x}}^{(i)}$, $L^{(i)}(1) = \bar{\boldsymbol{y}}^{(i)}$. $\mathcal{P}$ computes a univariate polynomial $q^{(i)}(X)$ by restricting $\widetilde{W}_{i+1}$ to $L^{(i)}$, writing as $\sum_{j=0}^{k_{i+1}} q^{(i)}_j \cdot X^j$. $\mathcal{P}$ sends $(q^{(i)}_0 - \mu_{i,6k_{i+1}+1}, \ldots, q^{(i)}_{k_{i+1}} - \mu_{i,7k_{i+1}+1})$ to $\mathcal{V}$, and similarly, they obtain $([q^{(i)}_0], \ldots, [q^{(i)}_{k_{i+1}}])$, denoted by $[\![q^{(i)}(\cdot)]\!]$.
   (e) $\mathcal{V}$ selects $r^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$. $\mathcal{P}$ computes $m_{i+1} := q^{(i)}(r^{(i)})$. Then they set $\boldsymbol{r}_{i+1} := L^{(i)}(r^{(i)}) \in \mathbb{F}^{k_{i+1}}$, and compute $[m_{i+1}] := [q^{(i)}(r^{(i)})]$.
4. $\mathcal{P}$ and $\mathcal{V}$ perform the following checks.
   (a) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Lin` in Figure 2 on input tuples $\{([m_i], [\![g^{(i,j)}(\cdot)]\!], [\![h^{(i,j)}(\cdot)]\!])\}_{i\in[0,d),j\in[1,k_{i+1}]}$.
   (b) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Mult` in Figure 2 on input tuples $\{([\![q^{(i)}(\cdot)]\!], [\![h^{(i,k_{i+1})}(\cdot)]\!])\}_{i\in[0,d)}$.
   (c) $\mathcal{P}$ opens $[m_d] - \sum_{\omega\in\{0,1\}^{k_d}}[W_d(\omega)] \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d)$, where $\boldsymbol{\chi}_\omega(\cdot)$ is a Lagrange basis as defined in Def. 1. $\mathcal{V}$ checks whether it is a valid opening of $[0]$.
5. $\mathcal{V}$ accepts if and only $\mathcal{P}$ passes all the checks above. Otherwise, $\mathcal{V}$ rejects.

Fig. 3: Our ZK for layered arithmetic circuits in the $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$-hybrid model.

instance, given $[\mathbf{x}_1], \ldots, [\mathbf{x}_r]$, where $\mathbf{x}_1, \ldots, \mathbf{x}_r$ are over $\mathbb{F}_p$, the commitment of $\mathbf{x} := \sum_{i=1}^{r} \mathbf{x}_i \lambda_r$ over $\mathbb{F}_{p^r}$ can be computed from $[\mathbf{x}] := \sum_{i=1}^{r} \lambda_r \cdot [\mathbf{x}_i]$. We remark that there exist optimizations when considering concrete instantiations of subfield VOLE. As it is beyond the scope of paper, details of optimizations are omitted.

5 Interactive LPZK for General Arithmetic Circuits

In this section, we first describe an ILPZK proof system for proving the satisfiability of general arithmetic circuits, which satisfies all properties as indicated in Theorem 2. With this new ILPZK, we also construct a self-contained VOLE-based ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ in Figure 4. Finally, we prove that our protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ UC-realizes $\mathcal{F}_{\mathrm{ZK}}$ in the rVOLE-hybrid model with information-theoretic malicious security.

5.1 Our ILPZK construction

In contrast to layered circuits, gates of generic circuits may take inputs from all the previous layers. Due to this nature, it remained unclear over ten years how to adapt GKR to generic circuits without an $\mathcal{O}(d)$ overhead induced by arranging generic circuits into layered circuits. Virgo++ [36] is a recent breakthrough, extending GKR to generic circuits with linear prover time, and without $\mathcal{O}(d)$ overhead, from which we distill ideas. As usual, we first explicitly list notations used in this section here.

Notations. Let $\mathcal{C} : \mathbb{F}^{s_d} \to \mathbb{F}^{s_0}$ be a general circuit over $\mathbb{F}$ of depth d, size S, and fan-in two. We also label each layer of $\mathcal{C}$ from 0 to d, with 0 being the output layer and d being the input layer. Each layer $i \in [0, d)$ of $\mathcal{C}$ contains gates that each takes one input from layer $i+1$ and another input from previous layer j, where $j = i+1, \ldots, d$. Let $\mathbf{W}_i$ be the outputs of gates in layer $i \in [0, d]$ and define $s_i := |\mathbf{W}_i|$. Let $\mathbf{W}_{i,j}$ be the subset of outputs of gates in layer j that connect to layer i, and define $s_{i,j} := |\mathbf{W}_{i,j}|$, for $i \in [0, d)$, $j \in [i+1, d]$. By above definitions, $S = \sum_{i=0}^{d-1} s_i$, and $s_j \geq s_{i,j}$ for all $j \in [1, d]$, $i \in [0, d)$. W.l.o.g., we always assume $s_{i,j} = 2^{k_{i,j}}$, and $s_i = 2^{k_i}$. Since each (add/mult) gate has only two inputs, there are at most $2s_i$ gates (from previous layers) connecting to gates in layer i, i.e., $2s_i \geq \sum_{j=i+1}^{d} s_{i,j}$. We always assume that $k_{i,i+1}$ is the largest among $\{k_{i,i+1}, \ldots, k_{i,d}\}$. We also re-define $\mathrm{add}_{i,j}, \mathrm{mult}_{i,j} : \{0,1\}^{k_i + k_{i,i+1} + k_{i,j}} \to \{0,1\}$, satisfying $\mathrm{add}_{i,j}(z, x, y) = 1$ ($\mathrm{mult}_{i,j}(z, x, y) = 1$) if and only if gate z is an addition (multiplication) gate in layer i (corresponds to $\mathbf{W}_i(z)$) that takes one input from gate x in layer $i+1$ (corresponds to $\mathbf{W}_{i,i+1}(x)$) and another input from gate y in layer j (corresponds to $\mathbf{W}_{i,j}(y)$).

We view each $\mathbf{W}_i \in \mathbb{F}^{s_i}$ ($\mathbf{W}_{i,j} \in \mathbb{F}^{s_{i,j}}$) as a function $W_i : \{0,1\}^{k_i} \to \mathbb{F}$ ($W_{i,j} : \{0,1\}^{k_{i,j}} \to \mathbb{F}$), and denote the multi-linear extension of $W_i, W_{i,j}, \mathrm{mult}_{i,j}, \mathrm{add}_{i,j}$ by $\widetilde{W_i}, \widetilde{W_{i,j}}, \widetilde{\mathrm{mult}}_{i,j}, \widetilde{\mathrm{add}}_{i,j}$, respectively. With above definitions, it holds for all

$i \in [0, d)$ that

$$\widetilde{W}_i(z) = \sum_{j=i+1}^{d} \sum_{x \in \{0,1\}^{k_{i,i+1}}} \sum_{y \in \{0,1\}^{k_{i,j}}} \Big(\widetilde{\mathrm{mult}}_{i,j}(z,x,y) \widetilde{W}_{i,i+1}(x) \widetilde{W}_{i,j}(y) + \widetilde{\mathrm{add}}_{i,j}(z,x,y) \big(\widetilde{W}_{i,i+1}(x) + \widetilde{W}_{i,j}(y) \big) \Big). \tag{5}$$

Protocol Overview. The ILPZK proof for general circuits shares the same bare-bone structure with that of Section 4, and now Eq.(5) is the key to the layer-by-layer reduction. Observe that for layer i, performing a sum-check on Eq.(5) would induce in total $d - i + 1$ sub-statements for the previous layers $i+1, \ldots, d$ (in contrast to 2 sub-statements for layer $i+1$ in the layered circuit setting). This in turn implies that when proceeding to layer i, there would be in total $i+1$ sub-statements from layers $0, \ldots, i-1$ (in contrast to 2 sub-statements from layer $i-1$ in the layered circuit setting). Therefore, a more sophisticated procedure of combining these sub-statements to one needs to be applied before sum-check.

The full construction also consists of d stages, and suppose in each stage $i \in [0, d)$, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ start with a commitment $[\widetilde{W}_i(\boldsymbol{r}_i)]$, where $\boldsymbol{r}_i \in \mathbb{F}^{k_i}$ is determined by $\mathcal{V}$. They run sum-check on Eq.(5) underneath commitments, reducing to $d-i+1$ sub-statements for previous layers $i+1, \ldots, d$. Then, they aggregate all sub-statements about layer $i+1$ via applying "sum-check" on another equation (will be explained later), obtaining a commitment on $\widetilde{W}_{i+1}(\boldsymbol{r}_{i+1})$. This allows $\mathcal{P}$ and $\mathcal{V}$ move to stage $i+1$.

From one statement to multiple sub-statements. Observe that directly perform "sum-check" on Eq.(5) would incur asymptotic overhead, since there are in total $k_{i,i+1} + \sum_{j=i+1}^{d} k_{i,j}$ variables to be summed over. To maintain a linear time prover, we rewrite Eq.(5) as in [36], by padding each y of length $k_{i,j}$ to $k_{i,i+1}$(as we assume $k_{i,i+1}$ is the largest among $\{k_{i,i+1}, \ldots, k_{i,d}\}$).

$$\begin{aligned}
\widetilde{W}_i(z) &= \sum_{x \in \{0,1\}^{k_{i,i+1}}} \sum_{j=i+1}^{d} \sum_{y \in \{0,1\}^{k_{i,j}}} \Big(\widetilde{\mathrm{mult}}_{i,j}(z,x,y)\ \widetilde{W}_{i,i+1}(x)\ \widetilde{W}_{i,j}(y) \\
&\quad + \widetilde{\mathrm{add}}_{i,j}(z,x,y) \big(\widetilde{W}_{i,i+1}(x) + \widetilde{W}_{i,j}(y) \big) \Big) \\
&= \sum_{x,y \in \{0,1\}^{k_{i,i+1}}} \sum_{j=i+1}^{d} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} y_l \Big) \cdot \Big(\widetilde{\mathrm{mult}}_{i,j}(z,x,y^{(i,j)}) \widetilde{W}_{i,i+1}(x) \widetilde{W}_{i,j}(y^{(i,j)}) \\
&\quad + \widetilde{\mathrm{add}}_{i,j}(z,x,y^{(i,j)}) \big(\widetilde{W}_{i,i+1}(x) + \widetilde{W}_{i,j}(y^{(i,j)}) \big) \Big),
\end{aligned} \tag{6}$$

where each $y^{(i,j)}$ refers to the first $k_{i,j}$ bits of a $y \in \{0,1\}^{k_{i,i+1}}$. Correctness of Eq.(6) follows from the fact that

$$\sum_{y \in \{0,1\}^{k_{i,j}}} f(y) = \sum_{y \in \{0,1\}^{k_{i,i+1}}} y_{k_{i,j}+1} \cdots y_{k_{i,i+1}} f(y^{(i,j)})$$

holds for any $f : \mathbb{F}^{k_{i,j}} \to \mathbb{F}$.

Then $\mathcal{P}$ and $\mathcal{V}$ apply a two-phase "sum-check" on Eq.(6) very similar to that in Section 4, which takes $\mathcal{O}\big(2^{k_{i,i+1}}\big) = \mathcal{O}(s_{i,i+1})$ computation. Details are deferred to Appendix C due to space constraint. In the end, they obtain sub-statements $[m_{i+1,i+1}], [m_{i,i+1}], \ldots, [m_{i,d}]$, where it is supposed to hold that $m_{i+1,i+1} = \widetilde{W}_{i+1,i+1}(\bar{\boldsymbol{x}}^{(i)})$ and $m_{i,j} = \widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)})$, for $j \in [i+1, d]$.

Aggregate multiple sub-statements to one statement. For simplicity, we define $\mathbf{W}_{i,i} = \mathbf{W}_{i-1,i}$ and $k_{i,i} = k_{i,i+1}$. Suppose for layer i, the $i+1$ sub-statements from above layers are $[m_{0,i}], \ldots, [m_{i,i}]$, where $m_{0,i} = \widetilde{W}_{0,i}(\bar{\boldsymbol{y}}^{(0,i)}), \ldots, m_{i-1,i} = \widetilde{W}_{i-1,i}(\bar{\boldsymbol{y}}^{(i-1,i)})$ and $m_{i,i} = \widetilde{W}_{i,i}(\bar{\boldsymbol{x}}^{(i-1)})$ all hold, and $\bar{\boldsymbol{y}}^{(0,i)}, \ldots, \bar{\boldsymbol{y}}^{(i-1,i)}, \bar{\boldsymbol{x}}^{(i-1)}$ are challenges from $\mathcal{V}$ in previous stages. The goal is to aggregate them to one statement for $\mathbf{W}_i$.

As $\mathbf{W}_{0,i}, \ldots, \mathbf{W}_{i,i}$ are subsets of $\mathbf{W}_i$, these $[m_{0,i}], \ldots, [m_{i,i}]$ can be computed from $\mathbf{W}_i$ and the previous challenges. Intuitively, this computation can be modeled as a layered arithmetic circuit $\mathcal{C}_i$ with private input $\mathbf{W}_i$ and output $(m_{0,i}, \ldots, m_{i,i})$, on which it suffices to apply original GKR. More concisely, observe that the evaluation of a multi-linear extension can be interpreted as simple as an inner product, e.g., $\widetilde{W}_{j,i}(\bar{\boldsymbol{y}}^{(j,i)}) = \sum_{\omega \in \{0,1\}^{k_{0,i}}} W_{j,i}(\omega) \cdot \boldsymbol{\chi}_\omega(\bar{\boldsymbol{y}}^{(j,i)})$, $j \in [0, i)$. So $\mathcal{C}_i$ essentially do the following things: select subsets of $\mathbf{W}_i$, compute expansions, and finally output the inner productions.

However, this conceptually simple approach would incur $\mathcal{O}(\log S)$ overhead in proof size, as computing expansions of $\bar{\boldsymbol{y}}^{(j,i)}$ requires circuits of depth $\mathcal{O}(k_{j,i})$. In fact, there exists a more efficient solution by fully exploiting the summation structure involved in the inner-product. Define $\texttt{EQ}_{j,i} : \{0,1\}^{k_i} \times \{0,1\}^{k_{j,i}} \to \mathbb{F}$, where $j \in [0, i)$, which takes as input a label z that indicates gates in layer i (corresponds to $\mathbf{W}_i(z)$), and a label y that indicates gates in layer i that connect to layer j (corresponds to $\mathbf{W}_{j,i}(y)$), outputs 1 if and only if they are exactly the same gate. The following holds:

$$
\begin{aligned}
m_{j,i} = \widetilde{W}_{j,i}(\bar{\boldsymbol{y}}^{(j,i)}) &= \sum_{\omega \in \{0,1\}^{k_{j,i}}} \widetilde{W}_{j,i}(\omega) \cdot \boldsymbol{\chi}_\omega(\bar{\boldsymbol{y}}^{(j,i)}) \\
&= \sum_{\omega \in \{0,1\}^{k_{j,i}}} \sum_{z \in \{0,1\}^{k_i}} \widetilde{W}_i(z) \cdot \texttt{EQ}_{j,i}(z, \omega) \cdot \boldsymbol{\chi}_\omega(\bar{\boldsymbol{y}}^{(j,i)}) \\
&= \sum_{z \in \{0,1\}^{k_i}} \widetilde{W}_i(z) \cdot \widetilde{\texttt{EQ}}_{j,i}(z, \bar{\boldsymbol{y}}^{(j,i)}).
\end{aligned} \tag{7}
$$

This allows to combine the sub-statements by taking a random linear combination on Eq.(7). Let $\mathcal{V}$ samples $\alpha^{(0,i)}, \ldots, \alpha^{(i,i)} \xleftarrow{\$} \mathbb{F}$, we have the following.

$$\begin{aligned}\underbrace{\sum_{j=0}^{i} \alpha^{(j,i)} m_{j,i}}_{m^{(i)}} &= \sum_{z \in \{0,1\}^{k_i}} \left(\alpha^{(i,i)} \widetilde{W}_i(z) \widetilde{\mathtt{EQ}}_{i-1,i}(z, \bar{\boldsymbol{x}}^{(i-1)}) + \sum_{j=0}^{i-1} \alpha^{(j,i)} \widetilde{W}_i(z) \widetilde{\mathtt{EQ}}_{j,i}(z, \bar{\boldsymbol{y}}^{(j,i)}) \right) \\ &= \sum_{z \in \{0,1\}^{k_i}} \widetilde{W}_i(z) \cdot \underbrace{\left(\alpha^{(i,i)} \widetilde{\mathtt{EQ}}_{i-1,i}(z, \bar{\boldsymbol{x}}^{(i-1)}) + \sum_{j=0}^{i-1} \alpha^{(j,i)} \widetilde{\mathtt{EQ}}_{j,i}(z, \bar{\boldsymbol{y}}^{(j,i)}) \right)}_{I^{(i)}(z)}\end{aligned} \tag{8}$$

Note that $I^{(i)}(z)$ only depends on the circuit topology and randomness selected by $\mathcal{V}$, it can be locally computed by each party. Therefore, it suffices for $\mathcal{P}$ and $\mathcal{V}$ to perform a "sum-check" on Eq.(8), and in the end, they would agree on a commitment $[m_i] := [\widetilde{W}_i(\bar{\boldsymbol{z}}^{(i)})]$, where $\bar{\boldsymbol{z}}^{(i)} \in \mathbb{F}^{k_i}$ is selected by $\mathcal{V}$. Details are deferred to Appendix C due to space constraint.

5.2 Complexity

Here we analyse the complexity of our ILPZK construction for general circuits.

Round complexity. Our construction can be divided into d stages, with each stage $i \in [1, d-2]$ having $2k_{i,i+1} + 1 + k_{i+1}$ rounds, stage 0 having $2k_{0,1} + 2 + k_1$ rounds, and stage $d-1$ having $2k_{d-1,d} + 2 + k_d$ rounds. Thus, there are $2 + \sum_{i=0}^{d-1}(k_{i,i+1} + 1 + k_{i+1}) = \mathcal{O}(d \log S)$ rounds in total.

Proof size. The proof size is the summation of length of sub-lines generated in each stage i, where $i \in [0, d)$. For each stage i, performing a two-phase sum-check on Eq.(6) and a sum-check on Eq.(8) incurs length of $6k_{i,i+1} + 3k_{i+1}$ in total, and committing to sub-statements incurs length of $d - i + 1$. Thus, the overall proof size is $\sum_{i=0}^{d-1}(6k_{i,i+1} + 3k_{i+1} + d - i + 1) = \mathcal{O}\big(n + d^2 + d \log S\big)$. We remark that here the $\mathcal{O}\big(d^2\big)$ term is always upper bounded by $\mathcal{O}(S)$. This is due to the fact that only if layer i connects to layer j, where $j < i$, then the sub-statement $[m_{j,i}]$ needs to be generated, yielding the number of sub-statements bounded by $2S$. This implies that only when the circuit is very narrow and almost every two layers are connected, then the proof size should be recognized as $\mathcal{O}(S)$.

Prover time. Applying "sum-check" dominates the prover time. By Lemma 2, for each stage i, performing a two-phase sum-check for generating sub-statements costs prover time $2\sum_{j=0}^{k_{i,i+1}} \mathcal{O}\big(2^{k_{i,i+1}-j}\big) = \mathcal{O}\big(2^{k_{i,i+1}}\big) = \mathcal{O}(s_{i,i+1})$, and performing a sum-check for combining sub-statements costs prover time $\sum_{j=0}^{k_{i+1}} \mathcal{O}\big(2^{k_{i+1}-j}\big) = \mathcal{O}\big(2^{k_{i+1}}\big) = \mathcal{O}(s_{i+1})$. Thus, it takes the prover overall $\mathcal{O}(S)$ time.

Verifier time. By a similar argument as in Section 4.2, the verifier $\mathcal{V}$ runs in time $\mathcal{O}\big(n + s_0 + d \log S + d^2 + T\big)$, where $\mathcal{O}(T)$ is the total time of evaluating $\widetilde{\mathrm{mult}}_{i,j}$, $\widetilde{\mathrm{add}}_{i,j}$ and $\widetilde{I}^{(i)}$. In general, $\mathcal{V}$ runs in time $\mathcal{O}(S)$.

5.3 Security Proof in UC-framework

We present in Figure 4 a self-contained ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ in the random VOLE-hybrid model, which is based on our ILPZK proof in Section 5.1 (and more details can be found in Appendix C). In addition, we explicitly present two sum-check like sub-protocols, one for generating sub-statements (Π_{SC1}, Figure 8), and the other for combining sub-statements (Π_{SC2}, Figure 9). We also design two tailored procedures for batch-checking linear and multiplicative constraints for this setting in Figure 10. In each stage i, where $i \in [0, d)$, protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ sequentially invokes Π_{SC1} and Π_{SC2}, proceeding from layer i to layer $i+1$. At the end of stage $d-1$, $\mathcal{P}$ and $\mathcal{V}$ perform checks on the degree-2 arithmetic constraints given by challenges from $\mathcal{V}$ and the circuit. We remark that $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ can be also extended to support any field, via building upon subfield VOLE.

The following theorem guarantees the security of our protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ with its proof deferred to Appendix B.2. Also, Theorem 4 implies Theorem 2.

Theorem 4. *Our ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ UC-realizes the ZK functionality $\mathcal{F}_{\mathrm{ZK}}$ in the $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$-hybrid model with information-theoretic malicious security. In particular, the environment $\mathcal{Z}$'s advantage is $\mathcal{O}\Big(\frac{d \log S}{|\mathbb{F}|}\Big)$.*

6 Acknowledgements

The authors would like to thank Yuval Ishai and Yanhong Xu for many helpful discussions of this work. We are also very grateful for the insightful comments from anonymous reviewers. The work was supported in part by the National Key Research and Development (R&D) Program of China under Grants 2020YFA0712300 and 2022YFA1004900 and in part by the National Natural Science Foundation of China under Grants 12031011, 12361141818, and 12101404.

Protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$

Notations follow Section 5.1. The prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that he holds a witness $\mathbf{w} \in \mathbb{F}^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

Offline phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ send (`Init`) to $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$, and $\mathcal{V}$ receives $\Delta \in \mathbb{F}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{Extend}, n + \sum_{i=0}^{d-1}(6k_{i,i+1} + (d-i+1) + 3k_{i+1}) + 1)$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$, which returns VOLE-based commitments on random values, denoted by $[\boldsymbol{\nu}], [\boldsymbol{\mu}], [\boldsymbol{\rho}], [\pi]$, where $\boldsymbol{\nu} \in \mathbb{F}^n$, $\boldsymbol{\mu}$ viewed as $\{\mu_{i,j}\}_{i\in[0,d),j\in[6k_{i,i+1}+d-i+1]}$, $\boldsymbol{\rho}$ viewed as $\{\rho_{i,j}\}_{i\in[1,d],j\in[3k_i]}$, and $\pi \in \mathbb{F}$.

Online phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ obtain $[\mathbf{w}]$ (by $\mathcal{P}$ sending $\boldsymbol{\delta} := \mathbf{w} - \boldsymbol{\nu}$ to $\mathcal{V}$).
2. For each layer i, $\mathcal{P}$ computes $\mathbf{W}_i$ and stores them. Note that for layer 0, $\mathcal{P}$ and $\mathcal{V}$ agree on $\mathbf{W}_0 := \mathbf{1} \in \mathbb{F}^{s_0}$. $\mathcal{V}$ picks a random $\boldsymbol{r}_0 \in \mathbb{F}^{k_0}$ and sends it to $\mathcal{P}$, then they compute $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$. For simplicity, we view m_0 as $[m_0]$ in the sense that $\mathcal{P}$ sets $M_{m_0} = 0$ and $\mathcal{V}$ sets $K_{m_0} = m_0 \cdot \Delta$.
3. For $i = 0, 1, \ldots, d-1$,
   (a) Generate sub-statements from layer i. $\mathcal{P}$ and $\mathcal{V}$ run $\Pi^{\mathcal{C}}_{\mathrm{SC1}}$ on input i, $[\boldsymbol{\mu}_i]$, $[m_i]$ and $\boldsymbol{r}_i$. They obtain $[m_{i+1,i+1}] := [\widetilde{W}_{i+1,i+1}(\bar{\boldsymbol{x}}^{(i)})]$ and $[m_{i,j}] := [\widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)})]$, where $j = i+1, \ldots, d$.
   (b) Combine sub-statements on layer $i+1$. $\mathcal{P}$ and $\mathcal{V}$ run $\Pi^{\mathcal{C}}_{\mathrm{SC2}}$ on input $i+1$, $[\boldsymbol{\rho}_{i+1}]$, $\bar{\boldsymbol{y}}^{(0,i+1)}, \ldots, \bar{\boldsymbol{y}}^{(i,i+1)}$, $\bar{\boldsymbol{x}}^{(i)}$ and $([m_{0,i+1}], \ldots, [m_{i,i+1}], [m_{i+1,i+1}])$. In the end, they obtain $[m_{i+1}] := [\widetilde{W}_{i+1}(\bar{\boldsymbol{z}}^{(i+1)})]$, and set $\boldsymbol{r}_{i+1} := \bar{\boldsymbol{z}}^{(i+1)}$.
4. $\mathcal{P}$ and $\mathcal{V}$ perform the following checks.
   (a) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Lin` in Figure 10 on input tuples $\{([m_i], [\![g^{(i,j)}(\cdot)]\!], [\![h^{(i,j)}(\cdot)]\!])\}_{i\in[0,d),j\in[1,k_{i,i+1}]}$ (from Step 3.a), and $\{([m^{(i)}], [\![I^{(i,j)}(\cdot)]\!])\}_{i\in[1,d],j\in[1,k_i]}$ (from Step 3.b).
   (b) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Mult` in Figure 10 on input tuples $\{([\![h^{(i,k_{i,i+1})}(\cdot)]\!], [m_{i+1,i+1}], [m_{i,i+1}], \ldots, [m_{i,d}])\}_{i\in[0,d)}$.
   (c) $\mathcal{P}$ opens $[m_d] - \sum_{\omega\in\{0,1\}^{k_d}}[W_d(\omega)] \cdot \boldsymbol{\chi}_\omega(\bar{\boldsymbol{z}}^{(d)})$, where $\boldsymbol{\chi}_\omega(\cdot)$ is a Lagrange basis as defined in Def. 1. $\mathcal{V}$ checks whether it is a valid opening of $[0]$.
5. $\mathcal{V}$ accepts if and only $\mathcal{P}$ passes all the checks above. Otherwise, $\mathcal{V}$ rejects.

Fig. 4: Our ZK for general arithmetic circuits in the $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$-hybrid model.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: CCS 2017. pp. 2087–2104. ACM (2017)
2. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-ole and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In: CRYPTO 2022. LNCS, vol. 13510, pp. 329–358. Springer (2022)
3. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In: CRYPTO 2023. LNCS, vol. 14085, pp. 581–615. Springer (2023)
4. Baum, C., Dittmer, S., Scholl, P., Wang, X.: Sok: vector ole-based zero-knowledge protocols. Des. Codes Cryptogr. **91**(11), 3527–3561 (2023)
5. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: CRYPTO 2021. LNCS, vol. 12828, pp. 92–122. Springer (2021)
6. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.14
7. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer (1993)
8. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: ASIACRYPT 2017. LNCS, vol. 10626, pp. 336–365. Springer (2017)
9. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier. In: EUROCRYPT 2022. LNCS, vol. 13276, pp. 275–304. Springer (2022)
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: CCS 2018. pp. 896–912. ACM (2018)
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: CCS 2019. pp. 291–308. ACM (2019)
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer (2019)
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001. pp. 136–145. IEEE Computer Society (2001)
14. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Goldwasser, S. (ed.) ITCS, 2012. pp. 90–112. ACM (2012)
15. Cramer, R., Damgård, I.: Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In: Krawczyk, H. (ed.) CRYPTO '98. vol. 1462, pp. 424–441. Springer (1998)
16. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving line-point zero knowledge: Two multiplications for the price of one. In: CCS 2022. pp. 829–841. ACM (2022)

17. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: ITC 2021. LIPIcs, vol. 199, pp. 5:1–5:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
18. Druk, E., Ishai, Y.: Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In: Naor, M. (ed.) Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014. pp. 169–182. ACM (2014)
19. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Dwork, C. (ed.) STOC 2008. pp. 113–122. ACM (2008)
20. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
21. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for R1CS. In: CRYPTO 2023. LNCS, vol. 14082, pp. 193–226. Springer (2023)
22. Lee, J., Setty, S., Thaler, J., Wahby, R.: Linear-time and post-quantum zero-knowledge snarks for r1cs. Cryptology ePrint Archive (2021), `https://eprint.iacr.org/2021/030`
23. Lin, F., Xing, C., Yao, Y.: More efficient zero-knowledge protocols over $\mathbb{Z}_{2^k}$ via galois rings. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX. Lecture Notes in Computer Science, vol. 14928, pp. 424–457. Springer (2024)
24. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)
25. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 129–140. Springer (1991)
26. Roy, L.: Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: CRYPTO 2022. LNCS, vol. 13507, pp. 657–687. Springer (2022)
27. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer (2020)
28. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. vol. 8043, pp. 71–89. Springer (2013)
29. Vu, V., Setty, S.T.V., Blumberg, A.J., Walfish, M.: A hybrid architecture for interactive verifiable computation. In: SP 2013. pp. 223–237. IEEE Computer Society (2013)
30. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zksnarks without trusted setup. In: SP 2018. pp. 926–943. IEEE Computer Society (2018)
31. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: IEEE Symposium on Security and Privacy 2021. pp. 1074–1091. IEEE (2021)
32. Weng, C., Yang, K., Yang, Z., Xie, X., Wang, X.: Antman: Interactive zero-knowledge proofs with sublinear communication. In: CCS 2022. pp. 2901–2914. ACM (2022)
33. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer (2019)

34. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO 2022. LNCS, vol. 13510, pp. 299–328. Springer (2022)
35. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: CCS 2021. pp. 2986–3001. ACM (2021)
36. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21. pp. 159–177. ACM (2021)
37. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: SP 2020. pp. 859–876. IEEE (2020)

Supplementary Material

A Missing Functionalities and Constructions

Security Model. In UC framework [13], security is defined via the comparison of an *ideal* world and a *real* world. In the *real* world, the parties interact with each other following the protocol. In the *ideal* world, the parties interact with an ideal functionality $\mathcal{F}$ that is designed to ideally realize the protocol rather than each other. There exists an environment $\mathcal{Z}$, who lives both in the *real* world and the *ideal* world, provides inputs to the parties and can read the outputs. The corrupted party $\mathcal{A}$, controlled by the environment $\mathcal{Z}$, interacts with honest parties in the *real* world. We consider active adversary and static corruption, namely the adversary $\mathcal{A}$'s behavior is arbitrary and not necessarily according to the protocol specification and corruption occurs before the protocol execution. Further, the environment $\mathcal{Z}$ is allowed to interact with $\mathcal{A}$ at any point throughout the protocol execution. The UC-security is guaranteed, if there is a simulator $\mathcal{S}$ plugged to the *ideal* world that interacts with $\mathcal{A}$ such that the environment $\mathcal{Z}$, who can observe $\mathcal{A}$'s view along with all parties' inputs and outputs, can not distinguish $\mathcal{S}$ and the honest parties. More formally, We say a protocol Π UC-realizes a functionality $\mathcal{F}$ with security parameter κ, if there is a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish the *ideal* world and the *real* world with advantage $1/\mathsf{poly}(\kappa)$.
Functionalities. We present the ideal ZK functionality in Figure 5, and the chosen-input VOLE functionality in Figure 6.

Functionality $\mathcal{F}_{\text{ZK}}$

Upon receiving $(\texttt{prove}, \mathcal{C}, \mathbf{w})$ from a prover $\mathcal{P}$ and $(\texttt{verify}, \mathcal{C})$ from a verifier $\mathcal{V}$, where the same circuit $\mathcal{C}$ is input by both parties, send $(\texttt{true})$ to $\mathcal{V}$ if $\mathcal{C}(\mathbf{w}) = \mathbf{1}$ and $(\texttt{false})$ otherwise.

Fig. 5: Functionality for zero-knowledge proofs for circuit satisfiability.

Sum-Check. We present the well-known sum-check protocol [24] in Figure 7.
More Details of LPZK. Security of our constructions relies on the following Schwartz-Zippel Lemma.

Lemma 3 (Schwartz-Zippel Lemma). *Let $f : \mathbb{F}^\ell \to \mathbb{F}$ be a non-zero ℓ-variate polynomial of total degree at most d, then*

$$\Pr\Big[f(\mathbf{x}) = 0 \mid \mathbf{x} \xleftarrow{\$} S^\ell\Big] \leq \frac{d}{|S|},$$

where S is an arbitrary set of $\mathbb{F}$.

[10] We remark that it is equivalent to receive a random Δ from honest $\mathcal{V}$.

Functionality $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$

Init: Upon receiving $(\texttt{Init}, N)$ from $P_{\mathcal{R}}$ and $(\texttt{Init}, N)$ from $P_{\mathcal{S}}$, sample $\Delta \xleftarrow{\$} \mathbb{F}$ if $P_{\mathcal{R}}$ is honest[10], and receive $\Delta \in \mathbb{F}$ from the adversary $\mathcal{A}$ otherwise. Send Δ to $\mathcal{V}$. Set $\texttt{counter} = 0$, and store Δ, N. All further $(\texttt{Init})$ commands will be ignored.

Extend: Upon receiving $(\texttt{Extend}, \mathbf{a}, \mathbf{b})$ from $P_{\mathcal{S}}$, where $\mathbf{a}, \mathbf{b} \in \mathbb{F}^{\ell}$, if $\texttt{counter} + \ell \leq N$, add $\texttt{counter}$ by ℓ and compute $\mathbf{v}_{\Delta} := \mathbf{a} \cdot \Delta + \mathbf{b}$. If $\texttt{counter} + \ell > N$, reserve the first $(N - \texttt{counter})$ entries of $\mathbf{a}, \mathbf{b}$, denoted by $\mathbf{a}', \mathbf{b}'$, and compute $\mathbf{v}_{\Delta} := \mathbf{a} \cdot \Delta + \mathbf{b}$. Send $\mathbf{v}_{\Delta}$ to $P_{\mathcal{R}}$. If $\texttt{counter} < N$, continue and wait for next $(\texttt{Extend})$ command from $P_{\mathcal{S}}$, otherwise halt.

Fig. 6: Ideal functionality for chosen-input VOLE over $\mathbb{F}$.

Protocol Sum-check

Given an ℓ-variate polynomial $f : \mathbb{F}^{\ell} \to \mathbb{F}$. Let $\deg_i(f)$ denote the degree of $f(X_1, \ldots, X_i, \ldots, X_\ell)$ in variable X_i. The protocol proceeds as follows.

- At the beginning, the prover $\mathcal{P}$ sends to the verifier $\mathcal{V}$ a value H claimed to equal the summation of f on the binary hypercube.
- In the first round, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_1(X_1)$ of degree at most $\deg_1(f)$ claimed to equal

$$\sum_{b_2, \ldots, b_\ell \in \{0,1\}} f(X_1, b_2, \ldots, b_\ell).$$

  $\mathcal{V}$ checks that $H = f_1(0) + f_1(1)$. $\mathcal{V}$ sends a random $r_1 \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
- In the i_{th} round, where $1 < i < \ell$, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_i(X_i)$ of degree at most $\deg_i(f)$, claimed to equal

$$\sum_{b_{i+1}, \ldots, b_\ell \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_\ell).$$

  $\mathcal{V}$ checks that $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$. $\mathcal{V}$ sends a random $r_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
- In the ℓ_{th} round, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_\ell(X_\ell)$ of degree at most $\deg_\ell(f)$, claimed to equal

$$f(r_1, \ldots, r_{\ell-1}, X_\ell).$$

  $\mathcal{V}$ checks that $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. Finally, $\mathcal{V}$ selects $r_\ell \xleftarrow{\$} \mathbb{F}$, and checks that $f(r_1, \ldots, r_\ell) = f_\ell(r_\ell)$. $\mathcal{V}$ will accept if and only if all the above checks pass. Otherwise, $\mathcal{V}$ rejects and aborts.

Fig. 7: Protocol for sum-check.

We first analyse the LPZK construction for proving linear constraints in Section 2.4. We give a deterministic extractor that extracts valid inputs $\mathbf{w}^* := (w_1^*, w_2^*)$ from the line $(\mathbf{a}^*, \mathbf{b}^*)$ generated by a malicious prover, and prove the extractor and protocol satisfy completeness, binding, soundness, and zero-knowledge. The extractor is simple: it reads off $\mathbf{w}^*$ as the first two entries of $\mathbf{a}^*$ if and only if the third entry of $\mathbf{a}^*$ is 0.

- **Completeness.** If the prover is honest, we have

$$\begin{aligned} v_3(\Delta) &= b_1 - \alpha \cdot b_2 \\ &= (a_1 - \alpha \cdot a_2 - \beta) \cdot \Delta + b_1 - \alpha \cdot b_2 \\ &= v_1(\Delta) - \alpha \cdot v_2(\Delta) - \beta \cdot \Delta \end{aligned}$$

  identically, as long as $a_1 = \alpha \cdot a_2 + \beta$.
- **Binding.** For any $\mathbf{w}' \neq \mathbf{w}^*$, the verifier's values $\mathbf{v}^*(\Delta)$ are consistent with $\mathbf{w}'$ only if $a_i^* \cdot \Delta + b_i^* = a_i' \cdot \Delta + b_i'$, for $i = 1, 2$. For any choice $(\mathbf{a}', \mathbf{b}') \neq (\mathbf{a}^*, \mathbf{b}^*)$ where these equality condition hold, there exists an index i with $a_i^* \neq a_i'$ and $b_i^* \neq b_i'$, and the prover can compute a corresponding guess $\Delta^* = (b_i^* - b_i')/(a_i^* - a_i')$. Since Δ is chosen uniformly at random, independent of the prover, the probability that $\Delta = \Delta^*$ is at most $1/|\mathbb{F}|$.
- **Soundness.** If the extracted input $\mathbf{w}^*$ does not satisfy that $a_1^* = \alpha \cdot a_2^* + \beta$, then the expression

$$v_1(x) - \alpha \cdot v_2(x) - \beta \cdot x - v_3(x) = (a_1^* - \alpha \cdot a_2^* - \beta) \cdot x + b_1^* - \alpha \cdot b_2^* - b_3^*$$

  is a non-trivial linear polynomial in x. This polynomial has only one root in $\mathbb{F}$, which gives a soundness error of at most $1/|\mathbb{F}|$.
- **Zero-knowledge.** $\mathcal{V}$ can simulate their view by generating v_1, v_2 uniformly at random, and computing $v_3 = v_1 - \alpha \cdot v_2 - \beta - \Delta$. We know that v_1, v_2 are uniformly random because of the uniform randomness of b_1, b_2, respectively.

Here we analyse the LPZK construction for proving multiplicative constraints in Section 2.4. Similarly, we give a deterministic extractor that extracts valid inputs $\mathbf{w}^* := (w_1^*, w_2^*, w_3^*)$ from the line $(\mathbf{a}^*, \mathbf{b}^*)$ generated by a malicious prover, and prove the extractor and protocol satisfy completeness, binding, soundness, and zero-knowledge. The extractor is simple: it reads off $\mathbf{w}^*$ as the first three entries of $\mathbf{a}^*$.

- **Completeness.** If the prover is honest, we have

$$\begin{aligned} v_4(\Delta) &= (b_1 a_2 + a_1 b_2 - b_3) \cdot \Delta + b_1 b_2 \\ &= (a_1 a_2 - a_3) \cdot \Delta^2 + (b_1 a_2 + a_1 b_2 - b_3) \cdot \Delta + b_1 b_2 \\ &= (a_1 \cdot \Delta + b_1) \cdot (a_2 \cdot \Delta + b_2) - (a_3 \cdot \Delta + b_3) \cdot \Delta \\ &= v_1(\Delta) \cdot v_2(\Delta) - v_3(\Delta) \cdot \Delta \end{aligned}$$

  identically, as long as $a_3 = a_1 \cdot a_2$.

– **Binding.** For any $\mathbf{w}' \neq \mathbf{w}^*$, the verifier's values $\mathbf{v}^*(\Delta)$ are consistent with $\mathbf{w}'$ only if $a_i^* \cdot \Delta + b_i^* = a_i' \cdot \Delta + b_i'$, for $i = 1, 2, 3$. For any choice $(\mathbf{a}', \mathbf{b}') \neq (\mathbf{a}^*, \mathbf{b}^*)$ where these equality condition hold, there exists an index i with $a_i^* \neq a_i'$ and $b_i^* \neq b_i'$, and the prover can compute a corresponding guess $\Delta^* = (b_i^* - b_i')/(a_i^* - a_i')$. Since Δ is chosen uniformly at random, independent of the prover, the probability that $\Delta = \Delta^*$ is at most $1/|\mathbb{F}|$.
– **Soundness.** If the extracted input $\mathbf{w}^*$ does not satisfy that $a_3^* = a_1^* \cdot a_2^*$, then the expression

$$v_1(x)v_2(x) - v_3(x)x - v_4(x) = (a_1^*a_2^* - a_3^*)x^2 + (b_1^*a_2^* + a_1^*b_2^* - b_3^* - a_4^*)x + b_1^*b_2^* - b_4^*$$

is a non-trivial degree-2 polynomial in x. This polynomial has at most two roots in $\mathbb{F}$, which gives a soundness error of at most $2/|\mathbb{F}|$.
– **Zero-knowledge.** $\mathcal{V}$ can simulate their view by generating v_1, v_2, v_3 uniformly at random, and computing $v_4 = v_1 v_2 - v_3 \cdot \Delta$. We know that v_1, v_2, v_3 are uniformly random because of the uniform randomness of b_1, b_2, b_3, respectively.

The above two checks admit LPZK constructions for checking arbitrary degree-2 constraints. Let $f(\mathbf{X})$ be an ℓ-variate polynomial of total degree-2 that captures a general degree-2 relation on $\mathbf{a}$, i.e., $f(\mathbf{a}) = 0$. This implies a degree-2 relation on $\mathbf{v} = \mathbf{a} \cdot x + \mathbf{b}$, written by $\hat{f}(\mathbf{v}) = f(\mathbf{a}) \cdot x^2 + f'(\mathbf{a}, \mathbf{b}) \cdot x + f''(\mathbf{a}, \mathbf{b})$, where $\hat{f}(\cdot)$ is obtained by multiplying x to the linear terms of $f(\cdot)$, and x^2 to the constant term of $f(\cdot)$, and $f'(\cdot), f''(\cdot)$ are degree-2 multi-variate polynomials determined by $f(\cdot)$. Then it suffices for $\mathcal{P}$ to define an additional entry with $v_{\ell+1}(x) := f'(\mathbf{a}, \mathbf{b}) \cdot x + f''(\mathbf{a}, \mathbf{b})$, and $\mathcal{V}$ can check by $\hat{f}(v_1(\Delta), \ldots, v_\ell(\Delta)) \stackrel{?}{=} v_{\ell+1}(\Delta)$, incurring a soundness error of at most $2/|\mathbb{F}|$ by Lemma 3.

B Missing Protocols & Proofs

B.1 Security Proof for Layered Circuits

Theorem 5 (Theorem 3, re-stated). *Our ZK protocol $\Pi_{\mathrm{ZKl}}^{\mathbb{F}}$ UC-realizes the ZK functionality $\mathcal{F}_{\mathrm{ZK}}$ (for proving satisfiability of layered circuits) in the $\mathcal{F}_{\mathrm{rVOLE}}^{\mathbb{F}}$-hybrid model with information-theoretic malicious security. In particular, the environment $\mathcal{Z}$'s advantage is $\mathcal{O}\left(\frac{d \log S}{|\mathbb{F}|}\right)$.*

Proof. ***Completeness.*** *Completeness directly follows our ILPZK construction in Section 4.1 and is omitted here.*
Security. *We divide our proof into two parts. First, we consider $\mathcal{P}$ is corrupted, then we consider $\mathcal{V}$ is corrupted. In each case, we build a* PPT *simulator $\mathcal{S}$ to interact with the corrupted party in the ideal world, such that the environment $\mathcal{Z}$ can distinguish the two worlds with advantage at most $\frac{5N+2d+3}{|\mathbb{F}|}$, where $N := \sum_{i=1}^{d} k_i = \mathcal{O}(d \log S)$.*
Corrupted $\mathcal{P}$: *$\mathcal{S}$ interacts with the adversary $\mathcal{A}$ as follows:*

1. *In the offline phase, $\mathcal{S}$ emulates $\mathcal{F}_{\mathrm{rVOLE}}^{\mathbb{F}}$ by recording $\mathbf{M}, \mathbf{x} \in \mathbb{F}^{n+d+1+7N}$ received from $\mathcal{A}$. Parse $\mathbf{M}$ as $\{\{M_{g_l^{(i,j)}}, M_{h_l^{(i,j)}}\}_{j\in[1,k_{i+1}],l\in[0,2]}, \{M_{q_l^{(i)}}\}_{l\in[0,k_{i+1}]}\}_{i\in[0,d)}$, $\{M_{w_i}\}_{i\in[1,n]}$ and M^*. Parse $\mathbf{x}$ as $\{\{x_{g_l^{(i,j)}}, x_{h_l^{(i,j)}}\}_{j\in[1,k_{i+1}],l\in[0,2]}, \{x_{q_l^{(i)}}\}_{l\in[0,k_{i+1}]}\}_{i\in[0,d)}$, $\{x_{w_i}\}_{i\in[1,n]}$ and x^*.*

2. *At the beginning of the online phase, upon receiving $\boldsymbol{\delta} \in \mathbb{F}^n$ from $\mathcal{A}$, $\mathcal{S}$ computes $\hat{w}_i := x_{w_i} + \delta_i$, for $i \in [1, n]$. Then $\mathcal{S}$ samples $\boldsymbol{r}_0 \xleftarrow{\$} \mathbb{F}^{k_0}$ and sends it to $\mathcal{A}$. As $\mathcal{C}$ is public, $\mathcal{S}$ sets $\mathbf{W}_0 := \mathbf{1} \in \mathbb{F}^{s_0}$ and computes $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$.*

3. *For stage $i \in [0, d)$, $\mathcal{S}$ proceeds as follows:*

   (a) *$\mathcal{S}$ samples $\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)} \xleftarrow{\$} \mathbb{F}^{k_{i+1}}$.*

   (b) *For $j = 1, \ldots, k_{i+1}$, upon receiving $\{\delta_{g_l^{(i,j)}}\}_{l=0,1,2}$ from $\mathcal{A}$, $\mathcal{S}$ sends $\bar{x}_j^{(i)}$ to $\mathcal{A}$. $\mathcal{S}$ computes $\hat{g}^{(i,j)}(\bar{x}_j^{(i)}) := \sum_{l=0}^{2}(x_{g_l^{(i,j)}} + \delta_{g_l^{(i,j)}}) \cdot (\bar{x}_j^{(i)})^l$, $\hat{g}^{(i,j)}(0) := x_{g_0^{(i,j)}} + \delta_{g_0^{(i,j)}}$, and $\hat{g}^{(i,j)}(1) := \sum_{l=0}^{2}(x_{g_l^{(i,j)}} + \delta_{g_l^{(i,j)}})$. Also $\mathcal{S}$ computes $\hat{M}_{g^{(i,j)}(\bar{x}_j^{(i)})} := \sum_{l=0}^{2} M_{g_l^{(i,j)}} \cdot (\bar{x}_j^{(i)})^l$, $\hat{M}_{g^{(i,j)}(0)} := M_{g_0^{(i,j)}}$, and $\hat{M}_{g^{(i,j)}(1)} := \sum_{l=0}^{2} M_{g_l^{(i,j)}}$.*

   (c) *For $j = 1, \ldots, k_{i+1}$, upon receiving $\{\delta_{h_l^{(i,j)}}\}_{l\in[0,2]}$ from $\mathcal{A}$, $\mathcal{S}$ sends $\bar{y}_j^{(i)}$ to $\mathcal{A}$. $\mathcal{S}$ computes $\hat{h}^{(i,j)}(\bar{y}_j^{(i)}) := \sum_{l=0}^{2}(x_{h_l^{(i,j)}} + \delta_{h_l^{(i,j)}}) \cdot (\bar{y}_j^{(i)})^l$, $\hat{h}^{(i,j)}(0) := x_{h_0^{(i,j)}} + \delta_{h_0^{(i,j)}}$, and $\hat{h}^{(i,j)}(1) := \sum_{l=0}^{2}(x_{h_l^{(i,j)}} + \delta_{h_l^{(i,j)}})$. Also $\mathcal{S}$ computes $\hat{M}_{h^{(i,j)}(\bar{y}_j^{(i)})} := \sum_{l=0}^{2} M_{h_l^{(i,j)}} \cdot (\bar{y}_j^{(i)})^l$, $\hat{M}_{h^{(i,j)}(0)} := M_{h_0^{(i,j)}}$, and $\hat{M}_{h^{(i,j)}(1)} := \sum_{l=0}^{2} M_{h_l^{(i,j)}}$.*

   (d) *Upon receiving $\{\delta_l^{(i)}\}_{l\in[0,k_{i+1}]}$ from $\mathcal{A}$, $\mathcal{S}$ samples $r^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{A}$. $\mathcal{S}$ defines the unique line $L^{(i)}$ such that $L^{(i)}(0) = \bar{\boldsymbol{x}}^{(i)}$ and $L^{(i)}(1) = \bar{\boldsymbol{y}}^{(i)}$. $\mathcal{S}$ computes $\boldsymbol{r}_{i+1} := L^{(i)}(r^{(i)}) \in \mathbb{F}^{k_{i+1}}$. $\mathcal{S}$ computes $\hat{q}^{(i)}(0) := \delta_0^{(i)} + x_{q_0^{(i)}}$, $\hat{q}^{(i)}(1) := \sum_{l=0}^{k_{i+1}}(\delta_l^{(i)} + x_{q_l^{(i)}})$, and $\hat{m}_{i+1} := \sum_{l=0}^{k_{i+1}}(\delta_l^{(i)} + x_{q_l^{(i)}}) \cdot (r^{(i)})^l$. Also, $\mathcal{S}$ computes $\hat{M}_{q^{(i)}(0)} := \hat{M}_{q_0^{(i)}}$, $\hat{M}_{q^{(i)}(1)} := \sum_{l=0}^{k_{i+1}} M_{q_l^{(i)}}$, and $\hat{M}_{m_{i+1}} := \sum_{l=0}^{k_{i+1}} M_{q_l^{(i)}} \cdot (r^{(i)})^l$.*

4. *Emulating the `Batch-Lin` procedure. There are $2N$ linear constraints. $\mathcal{S}$ samples $\boldsymbol{\lambda} \xleftarrow{\$} \mathbb{F}^{2N}$ and sends it to $\mathcal{A}$. $\mathcal{S}$ parses $\boldsymbol{\lambda}$ as $\{\lambda_{i,j}\}_{i\in[0,d-1],j\in[0,2k_{i+1}-1]}$,*

and computes $\hat{M}_{\mathrm{lin}}$ *as follows:*

$$\begin{aligned}\hat{M}_{\mathrm{lin}} :=& \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1}\lambda_{i,j}(\hat{M}_{g^{(i,j)}(\bar{x}_j^{(i)})} - \hat{M}_{g^{(i,j+1)}(0)} - \hat{M}_{g^{(i,j+1)}(1)})\\ &+\sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1}\lambda_{i,k_{i+1}+j}(\hat{M}_{h^{(i,j)}(\bar{y}_j^{(i)})} - \hat{M}_{h^{(i,j+1)}(0)} - \hat{M}_{h^{(i,j+1)}(1)})\\ &+\sum_{i=0}^{d-1}\lambda_{i,0}(\hat{M}_{m_i} - \hat{M}_{g^{(i,1)}(0)} - \hat{M}_{g^{(i,1)}(1)})\\ &+\sum_{i=0}^{d-1}\lambda_{i,k_{i+1}}(\hat{M}_{g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})} - \hat{M}_{h^{(i,1)}(0)} - \hat{M}_{h^{(i,1)}(1)}).\end{aligned}$$

Upon receiving M_{lin} *from* $\mathcal{A}$*,* $\mathcal{S}$ *checks that* $M_{\mathrm{lin}} \stackrel{?}{=} \hat{M}_{\mathrm{lin}}$*.* $\mathcal{S}$ *also computes* $\hat{x}_{\mathrm{lin}}$ *as follows:*

$$\begin{aligned}\hat{x}_{\mathrm{lin}} :=& \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1}\lambda_{i,j}(\hat{g}^{(i,j)}(\bar{x}_j^{(i)}) - \hat{g}^{(i,j+1)}(0) - \hat{g}^{(i,j+1)}(1))\\ &+\sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1}\lambda_{i,k_{i+1}+j}(\hat{h}^{(i,j)}(\bar{y}_j^{(i)}) - \hat{h}^{(i,j+1)}(0) - \hat{h}^{(i,j+1)}(1))\\ &+\sum_{i=0}^{d-1}\lambda_{i,0}(\hat{m}_i - \hat{g}^{(i,1)}(0) - \hat{g}^{(i,1)}(1))\\ &+\sum_{i=0}^{d-1}\lambda_{i,k_{i+1}}(\hat{g}^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)}) - \hat{h}^{(i,1)}(0) - \hat{h}^{(i,1)}(1)).\end{aligned}$$

$\mathcal{S}$ *checks that* $\hat{x}_{\mathrm{lin}} \stackrel{?}{=} 0$*.*

5. *Emulating the* **`Batch-Mult`** *procedure. There are* d *multiplicative constraints.* $\mathcal{S}$ *samples* $\boldsymbol{\alpha} \in \mathbb{F}^d$ *and sends it to* $\mathcal{A}$*,* $\mathcal{S}$ *computes* $\hat{x}_{\mathrm{mult}}, \hat{M}_{\mathrm{mult}}$ *as follows:*

$$\hat{M}_{\mathrm{mult}} := M^* + \sum_{i=0}^{d-1}\alpha_i \cdot \big(\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \cdot \hat{M}_{q^{(i)}(0)} \cdot \hat{M}_{q^{(i)}(1)}\big),$$

and

$$\begin{aligned}\hat{x}_{\mathrm{mult}} :=& x^* + \sum_{i=0}^{d-1}\alpha_i \cdot \Big(\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})\big(\hat{q}^{(i)}(0)\hat{M}_{q^{(i)}(0)} + \hat{q}^{(i)}(1)\hat{M}_{q^{(i)}(1)}\big)\\ &+ \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})(\hat{M}_{q^{(i)}(0)} + \hat{M}_{q^{(i)}(1)}) - \hat{M}_{h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)})}\Big).\end{aligned}$$

Upon receiving $(x_{\mathrm{mult}}, M_{\mathrm{mult}})$ *from* $\mathcal{A}$*,* $\mathcal{S}$ *checks that* $x_{\mathrm{mul}} \stackrel{?}{=} \hat{x}_{\mathrm{mult}}$ *and* $M_{\mathrm{mult}} \stackrel{?}{=} \hat{M}_{\mathrm{mult}}$*.*

6. *Emulating the opening of* $[0]$. $\mathcal{S}$ *computes*

$$\hat{M}_0 := \hat{M}_{m_d} - \sum_{\omega \in \{0,1\}^{k_d}} \hat{M}_{W_d(\omega)} \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d),$$
$$\hat{x}_0 := \hat{m}_d - \sum_{\omega \in \{0,1\}^{k_d}} \hat{W}_d(\omega) \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d),$$

$\mathcal{S}$ checks that $\hat{x}_0 \stackrel{?}{=} 0$. Upon receiving M_0 from $\mathcal{A}$, $\mathcal{S}$ checks that $\hat{M}_0 \stackrel{?}{=} M_0$.

7. *If $\mathcal{A}$ passes all the checks above, $\mathcal{S}$ sends $\hat{\mathbf{w}}$ to $\mathcal{F}_{\mathrm{ZK}}$.*

(I) *We first claim that if $\mathcal{A}$ passes the checks of $\mathcal{S}$, then an honest verifier $\mathcal{V}$ would always accept. In step 4, 5 and 6, one can view $\mathcal{S}$ is running an honest prover $\mathcal{P}$'s program to produce $\hat{M}_{\mathrm{lin}}, \hat{M}_{\mathrm{mult}}, \hat{x}_{\mathrm{mult}}, \hat{M}_0$ based on the previous generated transcripts. Therefore, checks of $\mathcal{S}$ guarantee that $M_{\mathrm{lin}}, M_{\mathrm{mult}}, x_{\mathrm{mult}}, M_0$ sent by $\mathcal{A}$ are honestly generated from these transcripts. By linear homomorphism of VOLE-based commitments, it should always hold that $\hat{K}_{\mathrm{lin}} = \hat{M}_{\mathrm{lin}} + \hat{x}_{\mathrm{lin}} \cdot \Delta$, $\hat{K}_{\mathrm{mult}} = \hat{M}_{\mathrm{mult}} + \hat{x}_{\mathrm{mult}} \cdot \Delta$, and $\hat{K}_0 = \hat{M}_0 + \hat{x}_0 \cdot \Delta$ for $\hat{K}_{\mathrm{lin}}, \hat{K}_{\mathrm{mult}}, \hat{K}_0$ computed by an honest $\mathcal{V}$ based on the same transcripts. Since x_{lin}, x_0 are supposed to be zero, x_{lin}, x_0 do not need to be transferred, and it suffices for $\mathcal{S}$ to check that $\hat{x}_{\mathrm{lin}} \stackrel{?}{=} 0$, and $x_0 \stackrel{?}{=} 0$. More specifically, we already show that if $M_{\mathrm{lin}} = \hat{M}_{\mathrm{lin}}, M_{\mathrm{mult}} = \hat{M}_{\mathrm{mult}}, x_{\mathrm{mult}} = \hat{x}_{\mathrm{mult}}$, $\hat{x}_{\mathrm{lin}} = 0$, $M_0 = \hat{M}_0$ and $\hat{x}_0 = 0$, then $\hat{K}_{\mathrm{lin}} = M_{\mathrm{lin}}, \hat{K}_0 = M_0$ and $\hat{K}_{\mathrm{mult}} = M_{\mathrm{mult}} + x_{\mathrm{mult}} \cdot \Delta$. This completes the claim.*

(II) *We consider the case that $\mathcal{A}$ passes the checks of $\mathcal{S}$ with $\hat{\mathbf{w}}$ such that $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$. Recall that we use two random linear combination checks to batch-check $2N$ linear constraints and d multiplicative constrains, respectively. By the well-known Schwartz-Zippel Lemma, if there is some tuple that dissatisfies the linear (multiplicative) constraint, then $\mathcal{A}$ will pass the linear (multiplicative) constraint batch-check of $\mathcal{S}$ with advantage at most $1/|\mathbb{F}|$. By a union bound, the random linear combination procedure increases the advantage of $\mathcal{A}$ by at most $2/|\mathbb{F}|$. From now on, we can assume that all the linear (multiplicative) constraints are satisfied, with $\hat{\mathbf{w}}$ satisfying $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$. This allows us to bound the soundness error in a very similar way to that in the GKR protocol [19], since $\mathcal{S}$ can be viewed as a special "GKR verifier"! We sketch the idea of GKR soundness analysis below.*

Suppose the prover of the GKR protocol begins with a false $\hat{\mathbf{w}}$ such that $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$, then the verifier will accept only if there is at least one round i in which the following occurs. The prover sends a univariate polynomial that differs from the prescribed polynomial but they agree on a random point later chosen by the verifier. Within the j_{th} invocation of the sum-check protocol in round i of the GKR protocol, the prover sends a degree-2 polynomial, thus he has advantage at most $2/|\mathbb{F}|$ by the Schwartz-Zippel Lemma. In the end of round i of the GKR protocol, the prover sends a degree-k_{i+1} polynomial, thus he has advantage at most $k_{i+1}/|\mathbb{F}|$. By a union bound, the soundness error of the GKR protocol is upper bounded by $\mathcal{O}(d \log |\mathcal{C}|/|\mathbb{F}|)$.

Let us turn back to our case. We first observe that given $\mathbf{M} \in \mathbb{F}^{n+d+1+7N}$*, those* $\hat{\mathbf{M}}$ *are determined by the randomness chosen by* $\mathcal{S}$*. Therefore,* $\mathcal{A}$ *has no advantage at fooling* $\mathcal{S}$ *about linear constraints on* $\hat{\mathbf{M}}$*. In addition, we note that* $\mathcal{S}$ *differs from an original GKR verifier in the last sum-check of each stage* i*. In more detail, the GKR verifier checks* $f_{\boldsymbol{r}_i}^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)}) \stackrel{?}{=} \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})q^{(i)}(0)q^{(i)}(1) + \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})(q^{(i)}(0) + q^{(i)}(1))$ *directly, while we instead apply a checking multiplicative constraints procedure implied by Eq.*(1)*. Again by the Schwartz-Zippel Lemma, this incurs a soundness error* $2/|\mathbb{F}|$ *in each stage (thus* $2d/|\mathbb{F}|$ *for* d *stages by a union bound). The advantage of* $\mathcal{A}$ *at fooling* $\mathcal{S}$ *about linear constraints on* $\hat{\mathbf{x}}$ *follows by the soundness of GKR protocol. Fooling* $\mathcal{S}$ *in each round* j *of stage* i*, where* $j \in [1, 2k_{i+1}]$ *(e.g., by providing* $g'^{(i,j)} \neq g^{(i,j)}$ *such that* $g'^{(i,j)}(\bar{x}_j^{(i)}) = g^{(i,j)}(\bar{x}_j^{(i)})$*) succeeds with probability at most* $\frac{2}{|\mathbb{F}|}$ *by the Schwartz-Zippel Lemma. In addition, fooling* $\mathcal{S}$ *by providing a* $q'^{(i)} \neq q^{(i)}$ *such that* $q'^{(i)}(r^{(i)}) = q^{(i)}(r^{(i)})$ *succeeds with probability at most* $\frac{k_{i+1}}{|\mathbb{F}|}$ *by the Schwartz-Zippel Lemma. These together lead to* $\mathcal{A}$*'s advantage at most* $5N/|\mathbb{F}|$*. By a union bound,* $\mathcal{A}$*'s advantage at this condition is upper bounded by* $(5N + 2d)/|\mathbb{F}|$*.*

(III) *Finally, we consider that* $\mathcal{A}$ *fails to pass the checks of* $\mathcal{S}$ *with* $\hat{\mathbf{w}}$ *such that* $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$*, but succeeds to pass the check of a honest verifier. This is essentially due to the statistical binding property of the VOLE-based commitments. Namely, in the opening phase,* $\mathcal{A}$ *can open a committed value* $[x]$ *to a value* $x' \neq x$ *with probability at most* $1/|\mathbb{F}|$ *by guessing* Δ *of* $\mathcal{V}$*.*

By a union bound, the environment $\mathcal{Z}$ *can distinguish between the real world and the ideal world with advantage at most* $(5N + 2d + 3)/|\mathbb{F}|$*.*

Corrupted $\mathcal{V}$***:*** *If* $\mathcal{S}$ *receives* `false` *from* $\mathcal{F}_{\mathrm{ZK}}$*, then it just aborts. Otherwise,* $\mathcal{S}$ *interacts with the adversary* $\mathcal{A}$ *as follows:*

1. *In the offline phase,* $\mathcal{S}$ *emulates* $\mathcal{F}_{\mathrm{rVOLE}}^{\mathbb{F}}$ *by recording* Δ *and* $\mathbf{K} \in \mathbb{F}^{n+d+1+7N}$ *received from* $\mathcal{A}$*. Parse* $\mathbf{K}$ *as* $\{\{K_{g_l^{(i,j)}}, K_{h_l^{(i,j)}}\}_{j\in[1,k_{i+1}],l\in[0,2]}, \{K_{q_l^{(i)}}\}_{l\in[0,k_{i+1}]}\}_{i\in[0,d-1]}$*,* $\{K_{w_i}\}_{i\in[1,n]}$ *and* K^**.*
2. *At the beginning of the online phase,* $\mathcal{S}$ *samples a random* $\boldsymbol{\delta} \stackrel{\$}{\leftarrow} \mathbb{F}^n$ *and sends it to* $\mathcal{A}$*, then* $\mathcal{S}$ *waits until receiving* $\boldsymbol{r}_0 \in \mathbb{F}^{k_0}$ *from* $\mathcal{A}$*. Note that as* $\mathcal{C}$ *is public,* $\mathcal{S}$ *sets* $\mathbf{W}_0 := \mathbf{1} \in \mathbb{F}^{s_0}$ *and computes* $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$*.* $\mathcal{S}$ *computes* $\hat{K}_{m_0} := m_0 \cdot \Delta$*, and* $\hat{K}_{w_i} := K_{w_i} + \delta_i \cdot \Delta$*, for* $i = 1, \ldots, n$*.*
3. *For* $i = 0, 1, \ldots, d-1$*,* $\mathcal{S}$ *proceeds as follows:*
   (a) $\mathcal{S}$ *picks random polynomials* $g^{(i,j)}(X_j) = \sum_{l=0}^{2} g_l^{(i,j)} \cdot X_j^l$ *and* $h^{(i,j)}(Y_j) = \sum_{l=0}^{2} h_l^{(i,j)} \cdot Y_j^l$*, by independently choosing their three coefficients uniformly at random, where* $j \in [1, k_{i+1}]$*.*
   (b) *For* $j = 1, \ldots, k_{i+1}$*,* $\mathcal{S}$ *sends* $g_0^{(i,j)}, g_1^{(i,j)}, g_2^{(i,j)}$ *to* $\mathcal{A}$*.* $\mathcal{S}$ *waits until receiving* $\bar{x}_j^{(i)}$ *from* $\mathcal{A}$*. Then* $\mathcal{S}$ *sets* $\hat{K}_{g^{(i,j)}(0)} := K_{g_0^{(i,j)}} + g_0^{(i,j)} \cdot \Delta$*,* $\hat{K}_{g^{(i,j}(1)} := \sum_{l=0}^{2} K_{g_l^{(i,j)}} + g_l^{(i,j)} \cdot \Delta$*, and* $\hat{K}_{g^{(i,j)}(\bar{x}_j^{(i)})} := \sum_{l=0}^{2} (K_{g_l^{(i,j)}} + g_l^{(i,j)} \cdot \Delta) \cdot (\bar{x}_j^{(i)})^l$*.*

(c) *For* $j = 1, \ldots, k_{i+1}$, $\mathcal{S}$ *sends* $h_0^{(i,j)}, h_1^{(i,j)}, h_2^{(i,j)}$ *to* $\mathcal{A}$. $\mathcal{S}$ *waits until receiving* $\bar{y}_j^{(i)}$ *from* $\mathcal{A}$. *Then* $\mathcal{S}$ *sets* $\hat{K}_{h^{(i,j)}(0)} := K_{h_0^{(i,j)}} + h_1^{(i,j)} \cdot \Delta$, $\hat{K}_{h^{(i,j)}(1)} := \sum_{l=0}^{2} K_{h_l^{(i,j)}} + h_l^{(i,j)} \cdot \Delta$, *and* $\hat{K}_{h^{(i,j)}(\bar{y}_j^{(i)})} := \sum_{l=0}^{2} (K_{h_l^{(i,j)}} + h_l^{(i,j)} \cdot \Delta) \cdot (\bar{y}_j^{(i)})^l$.

(d) $\mathcal{S}$ *picks a random univariate polynomial* $q^{(i)}(\cdot)$ *over* $\mathbb{F}$ *of degree at most* k_{i+1}, *by independently choosing its* $k_{i+1}+1$ *coefficients uniformly at random.* $\mathcal{S}$ *sends coefficients of* $q^{(i)}(\cdot)$ *(denoted by* $\{q_l^{(i)}\}_{l\in[0,k_{i+1}]}$*) to* $\mathcal{A}$. $\mathcal{S}$ *defines the unique line* $L^{(i)}$ *such that* $L^{(i)}(0) = \bar{\boldsymbol{x}}^{(i)}$ *and* $L^{(i)}(1) = \bar{\boldsymbol{y}}^{(i)}$. $\mathcal{S}$ *records* $r^{(i)} \in \mathbb{F}$ *received from* $\mathcal{A}$ *and computes* $\boldsymbol{r}_{i+1} := L^{(i)}(r^{(i)}) \in \mathbb{F}^{k_{i+1}}$. $\mathcal{S}$ *also computes* $\hat{K}_{q^{(i)}(0)} := K_{q_0^{(i)}} + q_0^{(i)} \cdot \Delta$, $\hat{K}_{q^{(i)}(1)} := \sum_{l=0}^{k_{i+1}} (K_{q_l^{(i)}} + q_l^{(i)} \cdot \Delta)$, $\hat{K}_{m_i} := \sum_{l=0}^{k_{i+1}} (K_{q_l^{(i)}} + q_l^{(i)} \cdot \Delta) \cdot (r^{(i)})^l$.

4. *Emulating the* `Batch-Lin` *procedure. There are* $2N$ *linear constraints. Upon receiving* $\boldsymbol{\lambda} \in \mathbb{F}^{2N}$ *from* $\mathcal{A}$, $\mathcal{S}$ *parses* $\boldsymbol{\lambda}$ *as* $\{\lambda_{i,j}\}_{i\in[0,d-1],j\in[0,2k_{i+1}-1]}$, *and computes* $\hat{K}_{\text{lin}}$ *as follows:*

$$\begin{aligned}
\hat{K}_{\text{lin}} := &\sum_{i=0}^{d-1} \sum_{j=1}^{k_{i+1}-1} \lambda_{i,j} (\hat{K}_{g^{(i,j)}(\bar{x}_j^{(i)})} - \hat{K}_{g^{(i,j+1)}(0)} - \hat{K}_{g^{(i,j+1)}(1)}) \\
&+ \sum_{i=0}^{d-1} \sum_{j=1}^{k_{i+1}-1} \lambda_{i,k_{i+1}+j} (\hat{K}_{h^{(i,j)}(\bar{y}_j^{(i)})} - \hat{K}_{h^{(i,j+1)}(0)} - \hat{K}_{h^{(i,j+1)}(1)}) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,0} (\hat{K}_{m_i} - \hat{K}_{g^{(i,1)}(0)} - \hat{K}_{g^{(i,1)}(1)}) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,k_{i+1}} (\hat{K}_{g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})} - \hat{K}_{h^{(i,1)}(0)} - \hat{K}_{h^{(i,1)}(1)})
\end{aligned}$$

Then $\mathcal{S}$ *sends* $\hat{K}_{\text{lin}}$ *to* $\mathcal{A}$.

5. *Emulating the* `Batch-Mult` *procedure. There are* d *linear constraints. Upon receiving* $\boldsymbol{\alpha} \in \mathbb{F}^d$ *from* $\mathcal{A}$, $\mathcal{S}$ *computes* $\hat{K}_{\text{mult}}$ *as follows:*

$$\begin{aligned}
\hat{K}_{\text{mult}} := K^* + \sum_{i=0}^{d-1} \alpha_i \cdot \Big(&\widetilde{\text{mult}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \hat{K}_{q^{(i)}(0)} \hat{K}_{q^{(i)}(1)} \\
&+ \big(\widetilde{\text{add}}_i(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})(\hat{K}_{q^{(i)}(0)} + \hat{K}_{q^{(i)}(1)}) - \hat{K}_{h^{(i,k_{i+1})}(\bar{y}_{k_{i+1}}^{(i)})}\big) \cdot \Delta \Big).
\end{aligned}$$

Then $\mathcal{S}$ *samples a random* $\hat{x}_{\text{mult}} \xleftarrow{\$} \mathbb{F}$ *and computes* $\hat{M}_{\text{mult}} := \hat{K}_{\text{mult}} - \hat{x}_{\text{mult}} \cdot \Delta$. *In the end,* $\mathcal{S}$ *sends* $(\hat{x}_{\text{mult}}, \hat{M}_{\text{mult}})$ *to* $\mathcal{A}$.

6. *Emulating the opening of* $[0]$. $\mathcal{S}$ *computes*

$$\hat{K}_0 := \hat{K}_{m_d} - \sum_{\omega\in\{0,1\}^{k_d}} \hat{K}_{W_d(\omega)} \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d).$$

Then $\mathcal{S}$ *sends* $\hat{K}_0$ *to* $\mathcal{A}$.

Recall that messages sent by an honest prover during the "commit" phases (i.e., step 1 and step 3 in $\Pi^{\mathbb{F}}_{\mathrm{ZK}l}$) are uniformly random as they are masked by $\mu_{i,j}$. Also, messages for performing multiplicative checks, i.e., $(x_{\mathrm{mult}}, M_{\mathrm{mult}})$, are uniformly random under the condition $K_{\mathrm{mult}} = x_{\mathrm{mult}} \cdot \Delta + M_{\mathrm{mult}}$. Besides, as for linear checks, $\hat{K}_{\mathrm{lin}}, \hat{K}_0$ from $\mathcal{S}$ are equal to M_{lin}, M_0 from an honest $\mathcal{P}$. Therefore, the above simulation is perfect. This completes the proof. □

B.2 Security Proof for General Circuits

Theorem 6 (Theorem 4, restated). *Our ZK protocol $\Pi^{\mathbb{F}}_{\mathrm{ZK}g}$ UC-realizes the ZK functionality $\mathcal{F}_{\mathrm{ZK}}$ in the $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$-hybrid model with information-theoretic malicious security. In particular, the environment $\mathcal{Z}$'s advantage is $\mathcal{O}\Big(\frac{d \log S}{|\mathbb{F}|}\Big)$.*

Proof. The proof shares many similarities with that of Theorem 3, and for completeness, we still provide a detailed proof here. ***Completeness.*** *Completeness follows our ILPZK construction in Section 5.1, and is omitted here.*
Security. *We divide our proof into two parts. First, we consider $\mathcal{P}$ is corrupted, then we consider $\mathcal{V}$ is corrupted. In each case, we build a* PPT *simulator $\mathcal{S}$ to interact with the corrupted party in the ideal world, such that the environment $\mathcal{Z}$ can distinguish the two worlds with advantage at most $\frac{4N_1+2N_2+2d+3}{|\mathbb{F}|}$, where $N_1 := \sum_{i=0}^{d-1} k_{i,i+1} = \mathcal{O}(d \log S)$ and $N_2 := \sum_{i=1}^{d} k_i = \mathcal{O}(d \log S)$*

Corrupted $\mathcal{P}$: *$\mathcal{S}$ interacts with the adversary $\mathcal{A}$ as follows:*

1. *In the offline phase, $\mathcal{S}$ emulates $\mathcal{F}^{\mathbb{F}}_{\mathrm{rVOLE}}$ by recording $\mathbf{M}, \mathbf{x} \in \mathbb{F}^{n+d(d+3)/2+1+6N_1+3N_2}$ received from $\mathcal{A}$. Parse $\mathbf{M}$ as $\Big\{\{M_{g_l^{(i,j)}}, M_{h_l^{(i,j)}}\}_{j\in[1,k_{i,i+1}]}, \{M_{I_l^{(i+1,j)}}\}_{j\in[1,k_{i+1}]}\Big\}_{i\in[0,d),l\in[0,2]}$, $\Big\{\{M_{m_{i,j}}\}_{j\in[i+1,d]}, \{M_{m_{i+1,i+1}}\}\Big\}_{i\in[0,d)}$, $\{M_{w_i}\}_{i\in[1,n]}$ and M^*. Parse $\mathbf{x}$ as x^*, $\Big\{\{x_{g_l^{(i,j)}}, x_{h_l^{(i,j)}}\}_{j\in[1,k_{i+1}]}, \{x_{I_l^{(i,j)}}\}_{j\in[1,k_{i+1}]}\Big\}_{i\in[0,d),l\in[0,2]}$, $\Big\{\{x_{m_{i,j}}\}_{j\in[i+1,d]}, \{x_{m_{i+1,i+1}}\}\Big\}_{i\in[0,d)}$, and $\{x_{w_i}\}_{i\in[1,n]}$.*
2. *At the beginning of the online phase, upon receiving $\boldsymbol{\delta} \in \mathbb{F}^n$ from $\mathcal{A}$, $\mathcal{S}$ computes $\hat{w}_i := x_{w_i} + \delta_i$, for $i \in [1, n]$. Then $\mathcal{S}$ samples $\boldsymbol{r}_0 \xleftarrow{\$} \mathbb{F}^{k_0}$ and sends it to $\mathcal{A}$. As $\mathcal{C}$ is public, $\mathcal{S}$ sets $\mathbf{W}_0 := \mathbf{1} \in \mathbb{F}^{s_0}$ and computes $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$.*
3. *For stage $i \in [0, d)$, $\mathcal{S}$ proceeds as follows:*
   (a) *$\mathcal{S}$ samples $\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)} \xleftarrow{\$} \mathbb{F}^{k_{i,i+1}}$, and $\bar{\boldsymbol{z}}^{(i+1)} \xleftarrow{\$} \mathbb{F}^{k_{i+1}}$.*
   (b) *For $j = 1, \ldots, k_{i,i+1}$, upon receiving $\{\delta_{g_l^{(i,j)}}\}_{l=0,1,2}$ from $\mathcal{A}$, $\mathcal{S}$ sends $\bar{x}_j^{(i)}$ to $\mathcal{A}$. $\mathcal{S}$ computes $\hat{g}^{(i,j)}(\bar{x}_j^{(i)}) := \sum_{l=0}^{2}(x_{g_l^{(i,j)}} + \delta_{g_l^{(i,j)}}) \cdot (\bar{x}_j^{(i)})^l$, $\hat{g}^{(i,j)}(0) := x_{g_0^{(i,j)}} + \delta_{g_0^{(i,j)}}$, and $\hat{g}^{(i,j)}(1) := \sum_{l=0}^{2}(x_{g_l^{(i,j)}} + \delta_{g_l^{(i,j)}})$. Also $\mathcal{S}$ computes $\hat{M}_{g^{(i,j)}(\bar{x}_j^{(i)})} := \sum_{l=0}^{2} M_{g_l^{(i,j)}} \cdot (\bar{x}_j^{(i)})^l$, $\hat{M}_{g^{(i,j)}(0)} := M_{g_0^{(i,j)}}$, and $\hat{M}_{g^{(i,j)}(1)} := \sum_{l=0}^{2} M_{g_l^{(i,j)}}$.*
   (c) *For $j = 1, \ldots, k_{i,i+1}$, upon receiving $\{\delta_{h_l^{(i,j)}}\}_{l\in[0,2]}$ from $\mathcal{A}$, $\mathcal{S}$ sends $\bar{y}_j^{(i)}$ to $\mathcal{A}$. $\mathcal{S}$ computes $\hat{h}^{(i,j)}(\bar{y}_j^{(i)}) := \sum_{l=0}^{2}(x_{h_l^{(i,j)}} + \delta_{h_l^{(i,j)}}) \cdot (\bar{y}_j^{(i)})^l$, $\hat{h}^{(i,j)}(0) := x_{h_0^{(i,j)}} + \delta_{h_0^{(i,j)}}$, and $\hat{h}^{(i,j)}(1) := \sum_{l=0}^{2}(x_{h_l^{(i,j)}} + \delta_{h_l^{(i,j)}})$. Also $\mathcal{S}$ computes*

$\hat{M}_{h^{(i,j)}(\bar{y}_j^{(i)})} := \sum_{l=0}^{2} M_{h_l^{(i,j)}} \cdot (\bar{y}_j^{(i)})^l$, $\hat{M}_{h^{(i,j)}(0)} := M_{h_0^{(i,j)}}$, *and* $\hat{M}_{h^{(i,j)}(1)} := \sum_{l=0}^{2} M_{h_l^{(i,j)}}$.

(d) *Upon receiving* $\delta_{m_{i+1,i+1}}, \delta_{i,i+1}, \ldots, \delta_{i,d}$ *from* $\mathcal{A}$, $\mathcal{S}$ *samples random* $\alpha^{(i+1,i+1)}$, $\alpha^{(i,i+1)}, \ldots, \alpha^{(i,d)} \xleftarrow{\$} \mathbb{F}$, *and sends them to* $\mathcal{A}$. $\mathcal{S}$ *computes* $\hat{m}_{i+1,i+1} := x_{m_{i+1,i+1}} + \delta_{m_{i+1,i+1}}$, *and* $\hat{m}_{i,i+1} := x_{m_{i,i+1}} + \delta_{m_{i,i+1}}, \ldots, \hat{m}_{i,d} := x_{m_{i,d}} + \delta_{m_{i,d}}$.

(e) *For* $j = 1, \ldots, k_{i+1}$, *upon receiving* $\{\delta_{I_l^{(i+1,j)}}\}_{l\in[0,2]}$ *from* $\mathcal{A}$, $\mathcal{S}$ *sends* $\bar{z}_j^{(i+1)}$ *to* $\mathcal{A}$. $\mathcal{S}$ *computes* $\hat{I}^{(i+1,j)}(\bar{z}_j^{(i+1)}) := \sum_{l=0}^{2}(x_{I_l^{(i+1,j)}} + \delta_{I_l^{(i+1,j)}}) \cdot (\bar{z}_j^{(i+1)})^l$, $\hat{I}^{(i+1,j)}(0) := x_{I_0^{(i+1,j)}} + \delta_{I_0^{(i+1,j)}}$, *and* $\hat{I}^{(i+1,j)}(1) := \sum_{l=0}^{2}(x_{I_l^{(i+1,j)}} + \delta_{I_l^{(i+1,j)}})$. *Also* $\mathcal{S}$ *computes* $\hat{M}_{I^{(i+1,j)}(\bar{z}_j^{(i+1)})} := \sum_{l=0}^{2} M_{I_l^{(i+1,j)}} \cdot (\bar{z}_j^{(i+1)})^l$, $\hat{M}_{I^{(i+1,j)}(0)} := M_{I_0^{(i+1,j)}}$, *and* $\hat{M}_{I^{(i+1,j)}(1)} := \sum_{l=0}^{2} M_{I_l^{(i+1,j)}}$.

(f) *Finally,* $\mathcal{S}$ *computes* $\hat{m}_{i+1} := \hat{I}^{(i+1,k_{i+1})}(\bar{z}_{k_{i+1}}^{(i+1)}) \cdot \big(I^{(i+1)}(\bar{z}_j^{(i+1)})\big)^{-1}$, *and* $\hat{M}_{m_{i+1}} := \hat{M}_{I^{(i+1,k_{i+1})}(\bar{z}_{k_{i+1}}^{(i+1)})} \cdot \big(I^{(i+1)}(\bar{\boldsymbol{z}}^{(i+1)})\big)^{-1}$.

4. *Emulating the* `Batch-Lin` *procedure. There are* $2N_1 + N_2$ *linear constraints.* $\mathcal{S}$ *samples* $\boldsymbol{\lambda} \xleftarrow{\$} \mathbb{F}^{2N_1+N_2}$ *and sends it to* $\mathcal{A}$. $\mathcal{S}$ *parses* $\boldsymbol{\lambda}$ *as* $\{\lambda_{i,j}\}_{i\in[0,d),j\in[0,2k_{i,i+1}+k_{i+1})}$, *and computes* $\hat{M}_{\text{lin}}$ *as follows:*

$$
\begin{aligned}
\hat{M}_{\text{lin}} :=& \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1} \lambda_{i,j}\big(\hat{M}_{g^{(i,j)}(\bar{x}_j^{(i)})} - \hat{M}_{g^{(i,j+1)}(0)} - \hat{M}_{g^{(i,j+1)}(1)}\big) \\
&+ \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1} \lambda_{i,k_{i,i+1}+j}\big(\hat{M}_{h^{(i,j)}(\bar{y}_j^{(i)})} - \hat{M}_{h^{(i,j+1)}(0)} - \hat{M}_{h^{(i,j+1)}(1)}\big) \\
&+ \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1} \lambda_{i,j}\big(\hat{M}_{I^{(i+1,j)}(\bar{z}_j^{(i+1)})} - \hat{M}_{I^{(i+1,j+1)}(0)} - \hat{M}_{I^{(i+1,j+1)}(1)}\big) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,0}\big(\hat{M}_{m_i} - \hat{M}_{g^{(i,1)}(0)} - \hat{M}_{g^{(i,1)}(1)}\big) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,k_{i,i+1}}\big(\hat{M}_{g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})} - \hat{M}_{h^{(i,1)}(0)} - \hat{M}_{h^{(i,1)}(1)}\big) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,2k_{i,i+1}}\big(\big(\sum_{j=0}^{i+1} \alpha^{(j,i+1)} \cdot \hat{M}_{m_{j,i+1}}\big) - \hat{M}_{I^{(i+1,1)}(0)} - \hat{M}_{I^{(i+1,1)}(1)}\big).
\end{aligned}
$$

Upon receiving M_{lin} *from* $\mathcal{A}$, $\mathcal{S}$ *checks that* $M_{\text{lin}} \stackrel{?}{=} \hat{M}_{\text{lin}}$. $\mathcal{S}$ *also computes* $\hat{x}_{\text{lin}}$ *as follows:*

$$\begin{aligned}
\hat{x}_{\text{lin}} :=& \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1} \lambda_{i,j}(\hat{g}^{(i,j)}(\bar{x}_j^{(i)}) - \hat{g}^{(i,j+1)}(0) - \hat{g}^{(i,j+1)}(1)) \\
&+ \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1} \lambda_{i,k_{i,i+1}+j}(\hat{h}^{(i,j)}(\bar{y}_j^{(i)}) - \hat{h}^{(i,j+1)}(0) - \hat{h}^{(i,j+1)}(1)) \\
&+ \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1} \lambda_{i,j}(\hat{I}^{(i+1,j)}(\bar{z}_j^{(i+1)}) - \hat{I}^{(i+1,j+1)}(0) - \hat{I}^{(i+1,j+1)}(1)) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,0}(\hat{m}_i - \hat{g}^{(i,1)}(0) - \hat{g}^{(i,1)}(1)) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,k_{i,i+1}}(\hat{g}^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)}) - \hat{h}^{(i,1)}(0) - \hat{h}^{(i,1)}(1)) \\
&+ \sum_{i=0}^{d-1} \lambda_{i,2k_{i,i+1}}\big((\sum_{j=0}^{i+1} \alpha^{(j,i+1)} \cdot \hat{m}_{j,i+1}) - \hat{I}^{(i+1,1)}(0) - \hat{I}^{(i+1,1)}(1)\big).
\end{aligned}$$

$\mathcal{S}$ *checks that* $\hat{x}_{\text{lin}} \stackrel{?}{=} 0$.

5. *Emulating the* `Batch-Mult` *procedure. There are* d *multiplicative constraints. Let* $\boldsymbol{r}_i$ *denote* $\bar{\boldsymbol{z}}^{(i)} \in \mathbb{F}^{k_i}$ *for* $i \in [1, d]$. $\mathcal{S}$ *samples* $\boldsymbol{\beta} \in \mathbb{F}^d$ *and sends it to* $\mathcal{A}$, $\mathcal{S}$ *computes* $\hat{x}_{\text{mult}}, \hat{M}_{\text{mult}}$ *as follows:*

$$\hat{M}_{\text{mult}} := M^* + \sum_{i=0}^{d-1} \beta_i \Big(\sum_{j=i+1}^{d} \prod_{l=k_{i,j}+1}^{k_{i,i+1}} (\bar{y}_l^{(i)}) \big(\widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)}) \hat{M}_{m_{i+1,i+1}} \hat{M}_{m_{i,j}}\big)\Big),$$

and

$$\begin{aligned}
\hat{x}_{\text{mult}} :=& x^* + \sum_{i=0}^{d-1} \beta_i \cdot \Big(\sum_{j=i+1}^{d} \prod_{l=k_{i,j}+1}^{k_{i,i+1}} (\bar{y}_l^{(i)}) \cdot \\
&\big(\widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)}) \cdot \hat{m}_{i+1,i+1} \cdot \hat{m}_{i,j} \\
&+ \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)}) \cdot (\hat{m}_{i+1,i+1} + \hat{m}_{i,j})\big)\Big).
\end{aligned}$$

Upon receiving $(x_{\text{mult}}, M_{\text{mult}})$ *from* $\mathcal{A}$, $\mathcal{S}$ *checks that* $x_{\text{mul}} \stackrel{?}{=} \hat{x}_{\text{mult}}$ *and* $M_{\text{mult}} \stackrel{?}{=} \hat{M}_{\text{mult}}$.

6. *Emulating the opening of* $[0]$. $\mathcal{S}$ *computes*

$$\begin{aligned}
\hat{M}_0 :=& \hat{M}_{m_d} - \sum_{\omega \in \{0,1\}^{k_d}} \hat{M}_{W_d(\omega)} \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d), \\
\hat{x}_0 :=& \hat{m}_d - \sum_{\omega \in \{0,1\}^{k_d}} \hat{W}_d(\omega) \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d),
\end{aligned}$$

$\mathcal{S}$ *checks that* $\hat{x}_0 \stackrel{?}{=} 0$. *Upon receiving* M_0 *from* $\mathcal{A}$, $\mathcal{S}$ *checks that* $\hat{M}_0 \stackrel{?}{=} M_0$.

7. *If* $\mathcal{A}$ *passes all the checks above,* $\mathcal{S}$ *sends* $\hat{\mathbf{w}}$ *to* $\mathcal{F}_{\mathrm{ZK}}$.

(I) *We first claim that if* $\mathcal{A}$ *passes the checks of* $\mathcal{S}$, *then an honest verifier* $\mathcal{V}$ *would always accept. In step* 4, 5 *and* 6, *one can view* $\mathcal{S}$ *is running an honest prover* $\mathcal{P}$*'s program to produce* $\hat{M}_{\mathrm{lin}}, \hat{M}_{\mathrm{mult}}, \hat{x}_{\mathrm{mult}}, \hat{M}_0$ *based on the previous generated transcripts. Therefore, checks of* $\mathcal{S}$ *guarantee that* $M_{\mathrm{lin}}, M_{\mathrm{mult}}, x_{\mathrm{mult}}, M_0$ *sent by* $\mathcal{A}$ *are honestly generated from these transcripts. By linear homomorphism of VOLE-based commitments, it should always hold that* $\hat{K}_{\mathrm{lin}} = \hat{M}_{\mathrm{lin}} + \hat{x}_{\mathrm{lin}} \cdot \Delta$, $\hat{K}_{\mathrm{mult}} = \hat{M}_{\mathrm{mult}} + \hat{x}_{\mathrm{mult}} \cdot \Delta$, *and* $\hat{K}_0 = \hat{M}_0 + \hat{x}_0 \cdot \Delta$ *for* $\hat{K}_{\mathrm{lin}}, \hat{K}_{\mathrm{mult}}, \hat{K}_0$ *computed by an honest* $\mathcal{V}$ *based on the same transcripts. Since* x_{lin}, x_0 *are supposed to be zero,* x_{lin}, x_0 *does not need to be transferred, and it suffices for* $\mathcal{S}$ *to check that* $\hat{x}_{\mathrm{lin}} \stackrel{?}{=} 0$, *and* $x_0 \stackrel{?}{=} 0$. *More specifically, we already show that if* $M_{\mathrm{lin}} = \hat{M}_{\mathrm{lin}}, M_{\mathrm{mult}} = \hat{M}_{\mathrm{mult}}, x_{\mathrm{mult}} = \hat{x}_{\mathrm{mult}}$, $\hat{x}_{\mathrm{lin}} = 0$, $M_0 = \hat{M}_0$ *and* $\hat{x}_0 = 0$, *then* $\hat{K}_{\mathrm{lin}} = M_{\mathrm{lin}}, \hat{K}_0 = M_0$ *and* $\hat{K}_{\mathrm{mult}} = M_{\mathrm{mult}} + x_{\mathrm{mult}} \cdot \Delta$. *This completes the claim.*

(II) *We consider the case that* $\mathcal{A}$ *passes the checks of* $\mathcal{S}$ *with* $\hat{\mathbf{w}}$ *such that* $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$. *Recall that we use two random linear combination checks to batch-check* $2N_1 + N_2$ *linear constraints and* d *multiplicative constrains, respectively. By the well-known Schwartz-Zippel Lemma, if there is some tuple that dissatisfies the linear (multiplicative) constraint, then* $\mathcal{A}$ *will pass the linear (multiplicative) constraint batch-check of* $\mathcal{S}$ *with advantage at most* $1/|\mathbb{F}|$. *By a union bound, the random linear combination procedure increases the advantage of* $\mathcal{A}$ *by at most* $2/|\mathbb{F}|$. *From now on, we can assume that all the linear (multiplicative) constraints are satisfied, with* $\hat{\mathbf{w}}$ *satisfying* $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$. *This allows us to bound the soundness error in a very similar way to that in the GKR protocol [19], since* $\mathcal{S}$ *can be viewed as a special "GKR verifier"!*

In this case, we first observe that given $\mathbf{M} \in \mathbb{F}^{n+d(d+3)/2+1+6N_1+3N_2}$, *those* $\hat{\mathbf{M}}$ *are determined by the randomness chosen by* $\mathcal{S}$. *Therefore,* $\mathcal{A}$ *has no advantage at fooling* $\mathcal{S}$ *about linear constraints on* $\hat{\mathbf{M}}$. *In addition, we note that we essentially apply a checking multiplicative constraints procedure implied by Eq.*(6) *for each stage* i. *Again by the Schwartz-Zippel Lemma, this incurs a soundness error* $2/|\mathbb{F}|$ *in each stage (thus* $2d/|\mathbb{F}|$ *for* d *stages by a union bound). The advantage of* $\mathcal{A}$ *at fooling* $\mathcal{S}$ *about linear constraints on* $\hat{\mathbf{x}}$ *follows by the soundness of sum-check protocol. Since at each stage* i *of our protocol, there are* $2k_{i,i+1} + k_{i+1}$ *invocations of sum-check protocol, these together lead to* $\mathcal{A}$*'s advantage at most* $(4N_1 + 2N_2)/|\mathbb{F}|$. *By a union bound,* $\mathcal{A}$*'s advantage at this condition is upper bounded by* $(4N_1 + 2N_2 + 2d)/|\mathbb{F}|$.

(III) *Finally, we consider that* $\mathcal{A}$ *fails to pass the checks of* $\mathcal{S}$ *with* $\hat{\mathbf{w}}$ *such that* $\mathcal{C}(\hat{\mathbf{w}}) \neq \mathbf{1}$, *but succeeds to pass the check of a honest verifier. This is essentially due to the statistical binding property of the VOLE-based commitments. Namely, in the opening phase,* $\mathcal{A}$ *can open a committed value* $[x]$ *to a value* $x' \neq x$ *with probability at most* $1/|\mathbb{F}|$ *by guessing* Δ *of* $\mathcal{V}$.

By a union bound, the environment $\mathcal{Z}$ *can distinguish between the real world and the ideal world with advantage at most* $(4N_1 + 2N_2 + 2d + 3)/|\mathbb{F}|$.

Corrupted $\mathcal{V}$: *If $\mathcal{S}$ receives* `false` *from $\mathcal{F}_{\mathrm{ZK}}$, then it just aborts. Otherwise, $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ as follows:*

1. *In the offline phase, $\mathcal{S}$ emulates $\mathcal{F}_{\mathrm{rVOLE}}^{\mathbb{F}}$ by recording Δ and $\mathbf{K} \in \mathbb{F}^{n+d(d+3)/2+1+6N_1+3N_2}$ received from $\mathcal{A}$. Parse $\mathbf{K}$ as $\left\{\{K_{g_l^{(i,j)}}, K_{h_l^{(i,j)}}\}_{j\in[1,k_{i,i+1}]}, \{K_{I_l^{(i+1,j)}}\}_{j\in[1,k_{i+1}]}\right\}_{i\in[0,d),l\in[0,2]}$, $\left\{\{K_{m_{i,j}}\}_{j\in[i+1,d]}, \{K_{m_{i+1,i+1}}\}\right\}_{i\in[0,d)}$, $\{K_{w_i}\}_{i\in[1,n]}$ and M^**

2. *At the beginning of the online phase, $\mathcal{S}$ samples a random $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^n$ and sends it to $\mathcal{A}$, then $\mathcal{S}$ waits until receiving $\boldsymbol{r}_0 \in \mathbb{F}^{k_0}$ from $\mathcal{A}$. Note that as $\mathcal{C}$ is public, $\mathcal{S}$ sets $\mathbf{W}_0 := \mathbf{1} \in \mathbb{F}^{s_0}$ and computes $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$. $\mathcal{S}$ computes $\hat{K}_{m_0} := m_0 \cdot \Delta$, and $\hat{K}_{w_i} := K_{w_i} + \delta_i \cdot \Delta$, for $i = 1, \ldots, n$.*

3. *For $i = 0, 1, \ldots, d-1$, $\mathcal{S}$ proceeds as follows:*

   (a) *$\mathcal{S}$ picks random polynomials $g^{(i,j)}(X_j) = \sum_{l=0}^{2} g_l^{(i,j)} \cdot X_j^l$ and $h^{(i,j)}(Y_j) = \sum_{l=0}^{2} h_l^{(i,j)} \cdot Y_j^l$, by independently choosing their three coefficients uniformly at random, where $j \in [1, k_{i,i+1}]$. In addition, $\mathcal{S}$ picks random polynomials $I^{(i,j)}(Z_j) = \sum_{l=0}^{2} I_l^{(i,j)} \cdot Z_j^l$, by independently choosing their three coefficients uniformly at random, where $j \in [1, k_{i+1}]$.*

   (b) *For $j = 1, \ldots, k_{i,i+1}$, $\mathcal{S}$ sends $g_0^{(i,j)}, g_1^{(i,j)}, g_2^{(i,j)}$ to $\mathcal{A}$. $\mathcal{S}$ waits until receiving $\bar{x}_j^{(i)}$ from $\mathcal{A}$. Then $\mathcal{S}$ sets $\hat{K}_{g^{(i,j)}(0)} := K_{g_0^{(i,j)}} + g_0^{(i,j)} \cdot \Delta$, $\hat{K}_{g^{(i,j}(1)} := \sum_{l=0}^{2} K_{g_l^{(i,j)}} + g_l^{(i,j)} \cdot \Delta$, and $\hat{K}_{g^{(i,j)}(\bar{x}_j^{(i)})} := \sum_{l=0}^{2} (K_{g_l^{(i,j)}} + g_l^{(i,j)} \cdot \Delta) \cdot (\bar{x}_j^{(i)})^l$.*

   (c) *For $j = 1, \ldots, k_{i,i+1}$, $\mathcal{S}$ sends $h_0^{(i,j)}, h_1^{(i,j)}, h_2^{(i,j)}$ to $\mathcal{A}$. $\mathcal{S}$ waits until receiving $\bar{y}_j^{(i)}$ from $\mathcal{A}$. Then $\mathcal{S}$ sets $\hat{K}_{h^{(i,j)}(0)} := K_{h_0^{(i,j)}} + h_1^{(i,j)} \cdot \Delta$, $\hat{K}_{h^{(i,j)}(1)} := \sum_{l=0}^{2} K_{h_l^{(i,j)}} + h_l^{(i,j)} \cdot \Delta$, and $\hat{K}_{h^{(i,j)}(\bar{y}_j^{(i)})} := \sum_{l=0}^{2} (K_{h_l^{(i,j)}} + h_l^{(i,j)} \cdot \Delta) \cdot (\bar{y}_j^{(i)})^l$.*

   (d) *$\mathcal{S}$ samples $m_{i+1,i+1}, m_{i,i+1}, \ldots, m_{i,d} \xleftarrow{\$} \mathbb{F}$ and sends them to $\mathcal{A}$. $\mathcal{S}$ waits until receiving $\alpha^{(i+1,i+1)}, \alpha^{(i,i+1)}, \ldots, \alpha^{(i,d)}\mathbb{F}$ from $\mathcal{A}$. Then $\mathcal{S}$ sets $\hat{K}_{m_{i+1,i+1}} := K_{m_{i+1,i+1}} + m_{i+1,i+1} \cdot \Delta$ and $\hat{K}_{m_{i,j}} := K_{m_{i,j}} + m_{i,j} \cdot \Delta$ for $j \in [i+1, d]$.*

   (e) *For $j = 1, \ldots, k_{i+1}$, $\mathcal{S}$ sends $I_0^{(i+1,j)}, I_1^{(i+1,j)}, I_2^{(i+1,j)}$ to $\mathcal{A}$. $\mathcal{S}$ waits until receiving $\bar{z}_j^{(i+1)}$ from $\mathcal{A}$. Then $\mathcal{S}$ sets $\hat{K}_{I^{(i+1,j)}(0)} := K_{I_0^{(i+1,j)}} + I_1^{(i+1,j)} \cdot \Delta$, $\hat{K}_{I^{(i+1,j)}(1)} := \sum_{l=0}^{2} K_{I_l^{(i+1,j)}} + I_l^{(i+1,j)} \cdot \Delta$, and $\hat{K}_{I^{(i+1,j)}(\bar{z}_j^{(i+1)})} := \sum_{l=0}^{2} (K_{I_l^{(i+1,j)}} + I_l^{(i+1,j)} \cdot \Delta) \cdot (\bar{z}_j^{(i)})^l$.*

   (f) *Finally, $\mathcal{S}$ computes $\hat{K}_{m_{i+1}} := \hat{K}_{I^{(i+1,k_{i+1})}(\bar{z}_{k_{i+1}}^{(i+1)})} \cdot \left(I^{(i+1)}(\bar{\boldsymbol{z}}^{(i+1)})\right)^{-1}$.*

4. *Emulating the* `Batch-Lin` *procedure. There are $2N_1 + N_2$ linear constraints. Upon receiving $\boldsymbol{\lambda} \in \mathbb{F}^{2N_1+N_2}$ from $\mathcal{A}$, $\mathcal{S}$ parses $\boldsymbol{\lambda}$ as $\{\lambda_{i,j}\}_{i\in[0,d-1],j\in[0,2k_{i,i+1}+k_{i+1}-1]}$,*

and computes $\hat{K}_{\mathrm{lin}}$ *as follows:*

$$\begin{aligned}\hat{K}_{\mathrm{lin}} :=& \sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1}\lambda_{i,j}(\hat{K}_{g^{(i,j)}(\bar{x}_j^{(i)})} - \hat{K}_{g^{(i,j+1)}(0)} - \hat{K}_{g^{(i,j+1)}(1)})\\ &+\sum_{i=0}^{d-1}\sum_{j=1}^{k_{i,i+1}-1}\lambda_{i,k_{i,i+1}+j}(\hat{K}_{h^{(i,j)}(\bar{y}_j^{(i)})} - \hat{K}_{h^{(i,j+1)}(0)} - \hat{K}_{h^{(i,j+1)}(1)})\\ &+\sum_{i=0}^{d-1}\sum_{j=1}^{k_{i+1}-1}\lambda_{i,j}(\hat{K}_{I^{(i+1,j)}(\bar{z}_j^{(i+1)})} - \hat{K}_{I^{(i+1,j+1)}(0)} - \hat{K}_{I^{(i+1,j+1)}(1)})\\ &+\sum_{i=0}^{d-1}\lambda_{i,0}(\hat{K}_{m_i} - \hat{K}_{g^{(i,1)}(0)} - \hat{K}_{g^{(i,1)}(1)})\\ &+\sum_{i=0}^{d-1}\lambda_{i,k_{i,i+1}}(\hat{K}_{g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})} - \hat{K}_{h^{(i,1)}(0)} - \hat{K}_{h^{(i,1)}(1)})\\ &+\sum_{i=0}^{d-1}\lambda_{i,2k_{i,i+1}}\big((\sum_{j=0}^{i+1}\alpha^{(j,i+1)}\cdot\hat{K}_{m_{j,i+1}}) - \hat{K}_{I^{(i+1,1)}(0)} - \hat{K}_{I^{(i+1,1)}(1)}\big).\end{aligned}$$

Then $\mathcal{S}$ *sends* $\hat{K}_{\mathrm{lin}}$ *to* $\mathcal{A}$.

5. *Emulating the* `Batch-Mult` *procedure. There are* d *linear constraints. Let* $\boldsymbol{r}_i$ *denote* $\bar{\boldsymbol{z}}^{(i)} \in \mathbb{F}^{k_i}$ *for* $i \in [1,d]$. *Upon receiving* $\boldsymbol{\beta} \in \mathbb{F}^d$ *from* $\mathcal{A}$, $\mathcal{S}$ *computes* $\hat{K}_{\mathrm{mult}}$ *as follows:*

$$\begin{aligned}\hat{K}_{\mathrm{mult}} :=& K^* + \sum_{i=0}^{d-1}\beta_i\cdot\Big(\big(\sum_{j=i+1}^{d}\prod_{l=k_{i,j}+1}^{k_{i,i+1}}(\bar{y}_l^{(i)})\cdot\\ &(\widetilde{\mathrm{mult}}_{i,j}(\boldsymbol{r}_i,\bar{\boldsymbol{x}}^{(i)},\bar{\boldsymbol{y}}^{(i,j)})\cdot\hat{K}_{m_{i+1,i+1}}\cdot\hat{K}_{m_{i,j}}\\ &+\widetilde{\mathrm{add}}_{i,j}(\boldsymbol{r}_i,\bar{\boldsymbol{x}}^{(i)},\bar{\boldsymbol{y}}^{(i,j)})\cdot(\hat{K}_{m_{i+1,i+1}}+\hat{K}_{m_{i,j}})\cdot\Delta)\big)\\ &-\hat{K}_{h^{(i,k_{i,i+1})}(\bar{y}_{k_{i,i+1}}^{(i)})}\cdot\Delta\Big).\end{aligned}$$

Then $\mathcal{S}$ *samples a random* $\hat{x}_{\mathrm{mult}} \xleftarrow{\$} \mathbb{F}$ *and computes* $\hat{M}_{\mathrm{mult}} := \hat{K}_{\mathrm{mult}} - \hat{x}_{\mathrm{mult}}\cdot\Delta$. *In the end,* $\mathcal{S}$ *sends* $(\hat{x}_{\mathrm{mult}}, \hat{M}_{\mathrm{mult}})$ *to* $\mathcal{A}$.

6. *Emulating the opening of* $[0]$. $\mathcal{S}$ *computes*

$$\hat{K}_0 := \hat{K}_{m_d} - \sum_{\omega\in\{0,1\}^{k_d}}\hat{K}_{W_d(\omega)}\cdot\boldsymbol{\chi}_\omega(\boldsymbol{r}_d).$$

Then $\mathcal{S}$ *sends* $\hat{K}_0$ *to* $\mathcal{A}$.

Recall that messages sent by an honest prover during the "commit" phases (i.e., step 1 *and step* 3 *in* $\Pi^{\mathbb{F}}_{\mathrm{ZKg}}$*) are uniformly random as they are masked by* $\mu_{i,j}, \rho_{i,j}$. *Also, messages for performing multiplicative checks, i.e.,* $(x_{\mathrm{mult}}, M_{\mathrm{mult}})$, *are*

uniformly random under the condition $K_{\text{mult}} = x_{\text{mult}} \cdot \Delta + M_{\text{mult}}$. Besides, as for linear checks, $\hat{K}_{\text{lin}}, \hat{K}_0$ from $\mathcal{S}$ are equal to M_{lin}, M_0 from an honest $\mathcal{P}$. Therefore, the above simulation is perfect. This completes the proof. □

C Our ILPZK construction for general circuits.

The construction basically consists of three parts, generating sub-statements, combining sub-statements, and checking.

C.1 Generate sub-statements in stage i

In each stage i, where $i \in [0, d)$, suppose $\mathcal{P}$ and $\mathcal{V}$ start with a commitment $[m_i] := [\widetilde{W}_i(\bar{\boldsymbol{z}}^{(i)})]$, where $\bar{\boldsymbol{z}}^{(i)} \in \mathbb{F}^{k_i}$ is selected by $\mathcal{V}$. $\mathcal{P}$ first defines a $2k_{i,i+1}$-variate polynomial as follows

$$f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X}, \mathbf{Y}) := \sum_{j=i+1}^{d} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} Y_l \Big) \cdot \Big(\widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y}^{(i,j)}) \widetilde{W}_{i,i+1}(\mathbf{X}) \widetilde{W}_{i,j}(\mathbf{Y}^{(i,j)}) + \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y}^{(i,j)}) \big(\widetilde{W}_{i,i+1}(\mathbf{X}) + \widetilde{W}_{i,j}(\mathbf{Y}^{(i,j)}) \big) \Big),$$

where each $\mathbf{Y}^{(i,j)}$ refers to the first $k_{i,j}$ variables of $\mathbf{Y}$. Then $\mathcal{P}$ also defines two univariate polynomials

$$A_{\boldsymbol{r}_i}(\mathbf{X}) := \sum_{j=i+1}^{d} \sum_{y \in \{0,1\}^{k_{i,i+1}}} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} y_l \Big) \big(\widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, y^{(i,j)}) \widetilde{W}_{i,j}(y^{(i,j)}) + \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, y^{(i,j)}) \big),$$

and

$$B_{\boldsymbol{r}_i}(\mathbf{X}) := \sum_{j=i+1}^{d} \sum_{y \in \{0,1\}^{k_{i,i+1}}} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} y_l \Big) \cdot \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, y^{(i,j)}) \cdot \widetilde{W}_{i,j}(y^{(i,j)}).$$

With $f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X}, \mathbf{Y}), A_{\boldsymbol{r}_i}(x), B_{\boldsymbol{r}_i}(x)$ defined as above, we have the following

$$\sum_{x,y \in \{0,1\}^{k_{i,i+1}}} f^{(i)}_{\boldsymbol{r}_i}(x, y) = \sum_{x \in \{0,1\}^{k_{i,i+1}}} A_{\boldsymbol{r}_i}(x) \cdot \widetilde{W}_{i,i+1}(x) + B_{\boldsymbol{r}_i}(x).$$

For $j = 1, \ldots, k_{i,i+1}$, $\mathcal{P}$ defines a univariate polynomial

$$g^{(i,j)}(X_j) := \sum_{x_{j+1}, \ldots, x_{k_{i,i+1}} \in \{0,1\}} (A_{\boldsymbol{r}_i} \cdot \widetilde{W}_{i,i+1} + B_{\boldsymbol{r}_i})(\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_{j-1}, X_j, x_{j+1}, \ldots, x_{k_{i,i+1}}).$$

We have that

$$g^{(i,j)}(X_j) = \sum_{x_{j+1}, \ldots, x_{k_{i+1}} \in \{0,1\}} \sum_{y \in \{0,1\}^{k_{i+1}}} f^{(i)}_{\boldsymbol{r}_i}(\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_{j-1}, X_j, x_{j+1}, \ldots, x_{k_{i,i+1}}, y).$$

Similarly, for $j = 1, \ldots, k_{i,i+1}$, $\mathcal{P}$ defines a univariate polynomial

$$h^{(i,j)}(Y_j) := \sum_{y_{j+1},\ldots,y_{k_{i,i+1}} \in \{0,1\}} f^{(i)}_{\boldsymbol{r}_i}(\bar{\boldsymbol{x}}^{(i)}, \bar{y}^{(i)}_1, \ldots, \bar{y}^{(i)}_{j-1}, Y_j, y_{j+1}, \ldots, y_{k_{i,i+1}}).$$

The construction. The interaction of round j proceeds as follows, where $j \in [1, k_{i,i+1}]$,

- $\mathcal{P}$ computes $\mathbf{g}^{(i,j)} := (g_0^{(i,j)}, g_1^{(i,j)}, g_2^{(i,j)})$ and samples $\mathbf{b}^{(i,j)} \xleftarrow{\$} \mathbb{F}^3$. Then $\mathcal{P}$ sends $(\mathbf{g}^{(i,j)}, \mathbf{b}^{(i,j)})$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{VOLE}}$, which then returns $\mathbf{v}^{(i,j)}_{\Delta} := \mathbf{g}^{(i,j)} \cdot \Delta + \mathbf{b}^{(i,j)}$ to $\mathcal{V}$.
- Upon receiving $\mathbf{v}^{(i,j)}_{\Delta}$, $\mathcal{V}$ sends $\bar{x}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

The interaction of round $k_{i,i+1} + j$ proceeds as follows, where $j \in [1, k_{i,i+1}]$,

- $\mathcal{P}$ computes $\mathbf{h}^{(i,j)} := (h_0^{(i,j)}, h_1^{(i,j)}, h_2^{(i,j)})$ and samples $\mathbf{b}^{(i,k_{i,i+1}+j)} \xleftarrow{\$} \mathbb{F}^3$. Then $\mathcal{P}$ sends $(\mathbf{h}^{(i,j)}, \mathbf{b}^{(i,k_{i+1}+j)})$ to $\mathcal{F}^{\mathbb{F}}_{\mathrm{VOLE}}$, which then returns $\mathbf{v}^{(i,k_{i+1}+j)}_{\Delta} := \mathbf{h}^{(i,j)} \cdot \Delta + \mathbf{b}^{(i,k_{i+1}+j)}$ to $\mathcal{V}$.
- Upon receiving $\mathbf{v}^{(i,k_{i+1}+j)}_{\Delta}$, $\mathcal{V}$ sends $\bar{y}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

The interaction of round $2k_{i,i+1} + 1$ proceeds as follows,

- $\mathcal{P}$ computes $m_{i+1,i+1} := \widetilde{W}_{i+1,i+1}(\bar{\boldsymbol{x}}^{(i)})$ and $m_{i,j} := \widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)})$, where $j \in [i+1, d]$. $\mathcal{P}$ samples $\mathbf{b}^{(i,2k_{i,i+1}+1)} \xleftarrow{\$} \mathbb{F}^{d-i+1}$. Then $\mathcal{P}$ sends $(\mathbf{m}^{(i)}, \mathbf{b}^{(i,2k_{i,i+1}+1)})$ to $\mathcal{F}_{\mathrm{VOLE}}$, which then returns to $\mathcal{V}$ a $\mathbf{v}^{(i,2k_{i+1}+1)}_{\Delta} := \mathbf{m}^{(i)} \cdot \Delta + \mathbf{b}^{(i,k_{i+1}+j)}$.
- Upon receiving $\mathbf{v}^{(i,2k_{i+1}+1)}_{\Delta}$, $\mathcal{V}$ sends $\alpha^{(i+1,i+1)}, \alpha^{(i,i+1)}, \ldots, \alpha^{(i,d)} \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

C.2 Combine sub-statements for layer $i+1$ in stage i

For layer $i+1$, there are $i+2$ sub-statements, $[m_{0,i+1}], \ldots, [m_{i,i+1}], [m_{i+1,i+1}]$, where $m_{j,i+1} = \widetilde{W}_{j,i+1}(\bar{\boldsymbol{y}}^{(j,i+1)})$ for $j \in [0, i]$ and $m_{i+1,i+1} = \widetilde{W}_{i+1,i+1}(\bar{\boldsymbol{x}}^{(i)})$. The prover $\mathcal{P}$ defines the k_{i+1}-variate polynomial as follows:

$$I^{(i+1)}(\mathbf{Z}) := \Big(\alpha^{(i+1,i+1)} \cdot \widetilde{\mathtt{EQ}}_{i,i+1}(\mathbf{Z}, \bar{\boldsymbol{x}}^{(i)}) + \sum_{j=0}^{i} \alpha^{(j,i+1)} \cdot \widetilde{\mathtt{EQ}}_{j,i+1}(\mathbf{Z}, \bar{\boldsymbol{y}}^{(j,i+1)})\Big).$$

For each $j \in [1, k_{i+1}]$, $\mathcal{P}$ computes a univariate polynomial of degree-2 as follows:

$$I^{(i+1,j)}(Z_j) := \sum_{z_{j+1},\ldots,z_{k_{i+1}} \in \{0,1\}} I^{(i+1)}(\bar{z}^{(i+1)}_1, \ldots, \bar{z}^{(i+1)}_{j-1}, Z_j, z_{j+1}, \ldots, z_{k_{i+1}}).$$

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ also compute $[m^{(i+1)}] := \sum_{j=0}^{i+1} \alpha^{(j,i+1)}[m_{j,i+1}]$.
The construction. The interaction of round $2k_{i,i+1} + 1 + j$ proceeds as follows, where $j \in [1, k_{i+1}]$,

– $\mathcal{P}$ computes $\mathbf{I}^{(i+1,j)} := (I_0^{(i+1,j)}, I_1^{(i+1,j)}, I_2^{(i+1,j)})$ and $\mathbf{b}^{(i,2k_{i,i+1}+1+j)} \xleftarrow{\$} \mathbb{F}^3$. Then $\mathcal{P}$ sends $(\mathbf{I}^{(i+1,j)}, \mathbf{b}^{(i,2k_{i,i+1}+1+j)})$ to $\mathcal{F}_{\text{VOLE}}^{\mathbb{F}}$, which then returns to $\mathcal{V}$ a $\mathbf{v}_{\Delta}^{(i,2k_{i+1}+1+j)} := \mathbf{I}^{(i+1,j)} \cdot \Delta + \mathbf{b}^{(i,2k_{i,i+1}+1+j)}$.
– Upon receiving $\mathbf{v}_{\Delta}^{(i,2k_{i+1}+1+j)}$, $\mathcal{V}$ sends $\bar{z}_j^{(i+1)} \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

In the end of round $2k_{i,i+1}+1+k_{i+1}$, they compute $[m_{i+1}] := [I^{(i+1,k_{i+1})}(\bar{z}_{k_{i+1}}^{(i+1)})] \cdot \big(I^{(i+1)}(\bar{\boldsymbol{z}}^{(i+1)})\big)^{-1}$, which is supposed to be $[\widetilde{W}_{i+1}(\bar{\boldsymbol{z}}^{(i+1)})]$.

C.3 Checking the constraints.

For each stage i, where $i \in [0, d)$, arithmetic constraints that the affine line $\mathbf{v}^{(i)}$ should satisfy are as follows.

$$[0] = [m_i] - [g^{(i,1)}(0)] - [g^{(i,1)}(1)],$$

and

$$[0] = [g^{(i,j-1)}(\bar{x}_{j-1}^{(i)})] - [g^{(i,j)}(0)] - [g^{(i,j)}(1)],$$

for $j = 2, \ldots, k_{i,i+1}$. And

$$[0] = [g^{(i,k_{i,i+1})}(\bar{x}_{k_{i,i+1}}^{(i)})] - [h^{(i,1)}(0)] - [h^{(i,1)}(1)],$$

and,

$$[0] = [h^{(i,j-1)}(\bar{y}_{j-1}^{(i)})] - [h^{(i,j)}(0)] - [h^{(i,j)}(1)],$$

for $j = 2, \ldots, k_{i,i+1}$. And

$$[0] = [m^{(i+1)}] - [I^{(i+1,1)}(0)] - [I^{(i+1,1)}(1)],$$

and

$$[0] = [I^{(i+1,j-1)}(\bar{z}_{j-1}^{(i+1)})] - [I^{(i+1,j)}(0)] - [I^{(i+1,j)}(1)],$$

for $j = 2, \ldots, k_{i+1}$. The above are all linear constraints, and there also exist multiplicative constraints. Given $[h^{(i,k_{i,i+1})}(\bar{y}_{k_{i,i+1}}^{(i)})]$ and $[m_{i+1,i+1}]$, $[m_{i,i+1}]$, $[m_{i,i+2}]$, …, $[m_{i,d}]$, it should hold that

$$\begin{aligned}
&h^{(i,k_{i,i+1})}(\bar{y}_{k_{i,i+1}}^{(i)}) = f_{\boldsymbol{r}_i}^{(i)}(\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)}) \\
&= \sum_{j=i+1}^{d} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} \bar{y}_l^{(i)} \Big) \cdot \Big(\widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)}) \cdot m_{i+1,i+1} \cdot m_{i,j} \\
&+ \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)}) \big(m_{i+1,i+1} + m_{i,j}\big) \Big) \\
&= \sum_{j=i+1}^{d} \alpha_{i,j} \cdot m_{i+1,i+1} \cdot m_{i,j} + \beta_{i,j} \cdot (m_{i+1,i+1} + m_{i,j}) \\
&= m_{i+1,i+1} \cdot \Big(\sum_{j=i+1}^{d} \alpha_{i,j} \cdot m_{i,j} + \beta_{i,j} \Big) + \Big(\sum_{j=i+1}^{d} \beta_{i,j} \cdot m_{i,j} \Big),
\end{aligned}$$

where $\alpha_{i,j} := (\prod_{l=k_{i,j}+1}^{k_{i,i+1}} \bar{y}_l^{(i)}) \cdot \widetilde{\mathrm{mult}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)})$ and $\beta_{i,j} := (\prod_{l=k_{i,j}+1}^{k_{i,i+1}} \bar{y}_l^{(i)}) \cdot \widetilde{\mathrm{add}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i,j)})$.

There remains a constraint for the witness. More concisely, it should hold that

$$[0] = [m_d] - \sum_{\omega \in \{0,1\}^{k_d}} [W_d(\omega)] \cdot \boldsymbol{\chi}_\omega(\bar{\boldsymbol{z}}^{(d)}),$$

where $[\mathbf{W}_d] := [\mathbf{w}]$ is generated at the beginning of stage 0. All degree-2 constraints can be guaranteed by an LPZK proof, increasing the dimension of $\mathbf{v}$ by a small constant. Note that the length for checking can be significantly compressed to 2 by taking a random linear combination on them.

Sub-protocol $\Pi_{\mathrm{SC1}}^{\mathcal{C}}$

Notations follow Section 5.1. Let i, $[\boldsymbol{\mu}_i]$, $[m_i]$ and $\boldsymbol{r}_i \in \mathbb{F}^{k_i}$ be the input, where $i \in [0, d)$. Note that we always assume $k_{i,i+1}$ is the largest among $\{k_{i,i+1}, \ldots, k_{i,d}\}$.

1. The prover $\mathcal{P}$ defines the $2k_{i,i+1}$-variate polynomial $f_{\boldsymbol{r}_i}^{(i)}(\mathbf{X}, \mathbf{Y})$ as follows:

$$f_{\boldsymbol{r}_i}^{(i)}(\mathbf{X}, \mathbf{Y}) := \sum_{j=i+1}^{d} \Big(\prod_{l=k_{i,j}+1}^{k_{i,i+1}} Y_l \Big) \cdot \Big(\widetilde{\mathrm{mult}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y}^{(i,j)}) \widetilde{W}_{i,i+1}(\mathbf{X}) \widetilde{W}_{i,j}(\mathbf{Y}^{(i,j)}) + \widetilde{\mathrm{add}}_{i,j}(\boldsymbol{r}_i, \mathbf{X}, \mathbf{Y}^{(i,j)}) \big(\widetilde{W}_{i,i+1}(\mathbf{X}) + \widetilde{W}_{i,j}(\mathbf{Y}^{(i,j)}) \big) \Big).$$

2. For $j = 1, \ldots, k_{i,i+1}$,
   (a) $\mathcal{P}$ computes a univariate polynomial $g^{(i,j)}(X_j)$ of degree-2, writing as $g_0^{(i,j)} + g_1^{(i,j)} \cdot X_j + g_2^{(i,j)} \cdot X_j^2$. $\mathcal{P}$ sends $g_0^{(i,j)} - \mu_{i,3j-2}$, $g_1^{(i,j)} - \mu_{i,3j-1}$, $g_2^{(i,j)} - \mu_{i,3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([g_0^{(i,j)}], [g_1^{(i,j)}], [g_2^{(i,j)}])$, denoted by $[\![g^{(i,j)}(\cdot)]\!]$.
   (b) $\mathcal{V}$ samples $\bar{x}_j^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
3. For $j = 1, \ldots, k_{i,i+1}$,
   (a) $\mathcal{P}$ computes a single variable polynomial $h^{(i,j)}(Y_j)$ of degree-2, writing as $h_0^{(i,j)} + h_1^{(i,j)} \cdot Y_j + h_2^{(i,j)} \cdot Y_j^2$. $\mathcal{P}$ sends $h_0^{(i,j)} - \mu_{i,3k_{i,i+1}+3j-2}$, $h_1^{(i,j)} - \mu_{i,3k_{i,i+1}+3j-1}$, $h_2^{(i,j)} - \mu_{i,3k_{i,i+1}+3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([h_0^{(i,j)}], [h_1^{(i,j)}], [h_2^{(i,j)}])$, denoted by $[\![h^{(i,j)}(\cdot)]\!]$.
   (b) $\mathcal{V}$ samples $\bar{y}_j^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
4. For each $j \in [i+1, d]$, let $\bar{\boldsymbol{y}}^{(i,j)} \in \mathbb{F}^{k_{i,j}}$ be the vector that contains the first $k_{i,j}$ entries of $\bar{\boldsymbol{y}}^{(i)}$. $\mathcal{P}$ commits to $\widetilde{W}_{i,i+1}(\bar{\boldsymbol{x}}^{(i)})$, and $\widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)})$ by sending $\widetilde{W}_{i,i+1}(\bar{\boldsymbol{x}}^{(i)}) - \mu_{i,6k_{i,i+1}+1}$, and $\widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)}) - \mu_{i,6k_{i,i+1}+1+j-i}$ to $\mathcal{V}$.
5. Output $[m_{i+1,i+1}] := [\widetilde{W}_{i+1,i+1}(\bar{\boldsymbol{x}}^{(i)})]$, $[m_{i,j}] := [\widetilde{W}_{i,j}(\bar{\boldsymbol{y}}^{(i,j)})]$, for $j \in [i+1, d]$.

Fig. 8: Sub-protocol for generating sub-statements in the rVOLE-hybrid model.

Sub-protocol $\Pi^{\mathcal{C}}_{\mathrm{SC2}}$

Notations follow Section 5.1. Let i, $[\boldsymbol{\rho_i}]$, $\boldsymbol{y}^{(0,i)}, \ldots, \boldsymbol{y}^{(i-1,i)}, \boldsymbol{x}^{(i-1)}$ and $([m_{0,i}], \ldots, [m_{i-1,i}], [m_{i,i}])$ be the input, where $i \in [1, d]$. Note that we always assume $k_{i,i+1}$ is the largest among $\{k_{i,i+1}, \ldots, k_{i,d}\}$.

1. The verifier $\mathcal{V}$ samples $\alpha^{(0,i)}, \ldots, \alpha^{(i,i)} \xleftarrow{\$} \mathbb{F}$, and sends them to $\mathcal{V}$.
2. The prover $\mathcal{P}$ defines the k_i-variate polynomial as follows:

$$I^{(i)}(\mathbf{Z}) := \Big(\alpha^{(i,i)} \cdot \widetilde{\mathtt{EQ}}_{i-1,i}(\mathbf{Z}, \bar{\boldsymbol{x}}^{(i-1)}) + \sum_{j=0}^{i-1} \alpha^{(j,i)} \cdot \widetilde{\mathtt{EQ}}_{j,i}(\mathbf{Z}, \bar{\boldsymbol{y}}^{(j,i)})\Big).$$

3. For $j = 1, \ldots, k_i$,
   (a) $\mathcal{P}$ computes a univariate polynomial of degree-2 as follows:

$$I^{(i,j)}(Z_j) := \sum_{z_{j+1}, \ldots, z_{k_i} \in \{0,1\}} I^{(i)}(\bar{z}^{(i)}_1, \ldots, \bar{z}^{(i)}_{j-1}, Z_j, z_{j+1}, \ldots, z_{k_i}).$$

   (b) Writing $I^{(i,j)}(Z_j) := I^{(i,j)}_0 + I^{(i,j)}_1 \cdot Z_j + I^{(i,j)}_2 \cdot Z_j^2$, and sends $I^{(i,j)}_0 - \rho_{i,3j-2}$, $I^{(i,j)}_1 - \rho_{i,3j-1}$, $I^{(i,j)}_2 - \rho_{i,3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([I^{(i,j)}_0], [I^{(i,j)}_1], [I^{(i,j)}_2])$, denoted by $[\![I^{(i,j)}(\cdot)]\!]$.
   (c) $\mathcal{V}$ samples $\bar{z}^{(i)}_j \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
4. Output $[m_i] := [I^{(i,k_i)}(\bar{z}^{(i)}_{k_i})] \cdot \big(I^{(i)}(\bar{\boldsymbol{z}}^{(i)})\big)^{-1}$, which is supposed to be $[\widetilde{W}_i(\bar{\boldsymbol{z}}^{(i)})]$.

Fig. 9: Sub-protocol for combining sub-statements in the rVOLE-hybrid model.

Procedure `Batch-Lin`: on input $\{[m_i], [\![g^{(i,j)}]\!], [\![h^{(i,j)}]\!]\}_{i\in[0,d),j\in[1,k_{i,i+1}]}$, and $\{([m^{(i)}], [\![I^{(i,j)}(\cdot)]\!])\}_{i\in[1,d],j\in[1,k_i]}$.

1. For $i \in [0,d)$,
   –for $j \in [1,k_{i,i+1}]$, $\mathcal{P}$ and $\mathcal{V}$ compute $([g^{(i,j)}(0)]$, $[g^{(i,j)}(1)]$, $[g^{(i,j)}(\bar{x}_j^{(i)})])$ and $([h^{(i,j)}(0)]$, $[h^{(i,j)}(1)]$, $[h^{(i,j)}(\bar{y}_j^{(i)})])$.
   –for $j \in [1,k_{i+1}]$, $\mathcal{P}$ and $\mathcal{V}$ compute $([I^{(i+1,j)}(0)]$, $[I^{(i+1,j)}(1)]$, $[I^{(i+1,j)}(\bar{z}_j^{(i+1)})])$.
   –The verifier $\mathcal{V}$ wants checks that $[0] \stackrel{?}{=} [m_i] - [g^{(i,1)}(0)] - [g^{(i,1)}(1)]$, $[0] \stackrel{?}{=} [g^{(i,k_{i+1})}(\bar{x}_{k_{i+1}}^{(i)})] - [h^{(i,1)}(0)] - [h^{(i,1)}(1)]$ and $[0] \stackrel{?}{=} [g^{(i,j)}(\bar{x}_j^{(i)})] - [g^{(i,j+1)}(0)] - [g^{(i,j+1)}(1)]$, $[0] \stackrel{?}{=} [h^{(i,j)}(\bar{y}_j^{(i)})] - [h^{(i,j+1)}(0)] - [h^{(i,j+1)}(1)]$ for $j \in [1,k_{i,i+1})$. And $[0] \stackrel{?}{=} [m^{i+1}] - [I^{(i,j+1)}(0)] - [I^{(i+1,j+1)}(1)]$, $[0] \stackrel{?}{=} [I^{(i+1,j)}(\bar{z}_j^{(i+1)})] - [I^{(i+1,j+1)}(0)] - [I^{(i+1,j+1)}(1)]$ for $j \in [1,k_{i+1})$. For simplicity, we naturally view these $2N_1+N_2$ constraints as $([x_i],[y_i],[z_i])$ such that $z_i = x_i + y_i$ should hold for all $i \in [N]$, where $N_1 := \sum_{i=0}^{d-1} k_{i,i+1}$ and $N_2 := \sum_{i=0}^{d-1} k_{i+1}$.
2. $\mathcal{V}$ samples $\boldsymbol{\lambda} \xleftarrow{\$} \mathbb{F}^{2N_1+N_2}$, and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $M_{\text{lin}} := \sum_{i=1}^{2N_1+N_2} \lambda_i (M_{z_i} - M_{x_i} - M_{y_i})$, and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ computes $K_{\text{lin}} := \sum_{i=1}^{2N_1+N_2} \lambda_i (K_{z_i} - K_{x_i} - K_{y_i})$, and checks that $K_{\text{lin}} \stackrel{?}{=} M_{\text{lin}}$.

Procedure `Batch-Mult`: on input $\{([\![h^{(i,k_{i,i+1})}(\cdot)]\!], [m_{i+1,i+1}], [m_{i,i+1}], \ldots, [m_{i,d}])\}_{i\in[0,d)}$.

1. For $i \in [0,d)$, $\mathcal{P}$ and $\mathcal{V}$ compute $[z_i] := [h^{(i,k_{i,i+1})}(\bar{y}_{k_{i,i+1}}^{(i)})]$, $[x_i] := [m_{i+1,i+1}]$, $[y_{i,j}] := [m_{i,j}]$, and $a_{i,j} := (\prod_{l=k_{i,j}+1}^{k_{i,i+1}}) \cdot \widetilde{\text{mult}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$, $b_{i,j} := (\prod_{l=k_{i,j}+1}^{k_{i,i+1}}) \cdot \widetilde{\text{add}}_{i,j}(\boldsymbol{r}_i, \bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{y}}^{(i)})$, where $j = i+1, \ldots, d$.
2. $\mathcal{V}$ wants to check that $z_i \stackrel{?}{=} \sum_{j=i+1}^{d} a_{i,j} x_i y_{i,j} + b_{i,j}(x_i + y_{i,j})$ for all i.
3. $\mathcal{V}$ samples $\boldsymbol{\beta} \xleftarrow{\$} \mathbb{F}^d$ and sends it to $\mathcal{P}$.
4. They consume a random VOLE correlation $[\pi]$ in the sense that $\mathcal{P}$ computes $M_{\text{mult}} := M_\pi + \sum_{i=0}^{d-1} \beta_i (\sum_{j=i+1}^{d} a_{i,j} M_{x_i} M_{y_{i,j}})$, and

$$x_{\text{mult}} := \pi + \sum_{i=0}^{d-1} \beta_i \Big(\big(\sum_{j=i+1}^{d} a_{i,j}(y_{i,j} M_{x_i} + x_i M_{y_{i,j}}) + b_{i,j}(M_{x_i} + M_{y_{i,j}}) \big) - M_{z_i} \Big),$$

   while $\mathcal{V}$ computes

$$K_{\text{mult}} := K_\pi + \sum_{i=0}^{d-1} \beta_i \Big(\big(\sum_{j=i+1}^{d} a_{i,j} K_{x_i} K_{y_{i,j}} + b_{i,j}(K_{x_i} + K_{y_{i,j}}) \big) - K_{z_i} \cdot \Delta \Big).$$

5. $\mathcal{P}$ sends $(x_{\text{mult}}, M_{\text{mult}})$ to $\mathcal{V}$, who then checks that $K_{\text{mult}} \stackrel{?}{=} M_{\text{mult}} + x_{\text{mult}} \cdot \Delta$.

Fig. 10: Procedures for batch-checking linear and multiplicative constraints.

D VOLE-based ZK for circuits over a small field $\mathbb{F}_p$.

Functionality $\mathcal{F}_{\text{sVOLE}}^{\mathbb{F}_p,\mathbb{F}_{p^r}}$

Init: Upon receiving (`Init`) from both parties, sample $\Delta \xleftarrow{\$} \mathbb{F}_{p^r}$ if $P_{\mathcal{R}}$ is honest, and receive $\Delta \in \mathbb{F}_{p^r}$ from the adversary $\mathcal{A}$ otherwise. Store Δ and send it to $P_{\mathcal{R}}$. All further (`Init`) commands will be ignored.

Extend-Subfield: Upon receiving (`Extend-Subfield`, N) from both parties, proceed as follows:

1. If $P_{\mathcal{R}}$ is honest, sample $\mathbf{K} \xleftarrow{\$} \mathbb{F}_{p^r}^N$. Otherwise receive $\mathbf{K}$ from $\mathcal{A}$.
2. If $P_{\mathcal{S}}$ is honest, sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}_p^N$ and compute $\mathbf{M} := \mathbf{K} - \Delta \cdot \mathbf{x} \in \mathbb{F}_{p^r}^N$. Otherwise, receive $\mathbf{x} \in \mathbb{F}_p^N$ and $\mathbf{M} \in \mathbb{F}_{p^r}^N$ from $\mathcal{A}$ and then recompute $\mathbf{K} := \mathbf{M} + \Delta \cdot \mathbf{x}$.
3. Send $(\mathbf{x}, \mathbf{M})$ to $P_{\mathcal{S}}$ and $\mathbf{K}$ to $P_{\mathcal{R}}$.

Extend: Upon receiving (`Extend`, N) from both parties, proceed as follows:

1. If $P_{\mathcal{R}}$ is honest, sample $\mathbf{K} \xleftarrow{\$} \mathbb{F}^N$. Otherwise receive $\mathbf{K}$ from $\mathcal{A}$.
2. If $P_{\mathcal{S}}$ is honest, sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}^N$ and compute $\mathbf{M} := \mathbf{K} - \Delta \cdot \mathbf{x} \in \mathbb{F}^N$. Otherwise, receive $\mathbf{x} \in \mathbb{F}^N$ and $\mathbf{M} \in \mathbb{F}^N$ from $\mathcal{A}$ and then recompute $\mathbf{K} := \mathbf{M} + \Delta \cdot \mathbf{x}$.
3. Send $(\mathbf{x}, \mathbf{M})$ to $P_{\mathcal{S}}$ and $\mathbf{K}$ to $P_{\mathcal{R}}$.

Fig. 11: Ideal functionality for random subfield VOLE.

Protocol $\Pi^{\mathbb{F}_p}_{\mathrm{ZK}l}$

Notations follow Section 4.1. The prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that he holds a witness $\mathbf{w} \in \mathbb{F}_p^n$ such that $\mathcal{C}(\mathbf{w}) = \mathbf{1}$.

Offline phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ send (`Init`) to $\mathcal{F}_{\mathrm{sVOLE}}^{\mathbb{F}_p,\mathbb{F}_{p^r}}$, and $\mathcal{V}$ receives $\Delta \in \mathbb{F}_{p^r}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send (`Extend-Subfield`, n) to $\mathcal{F}_{\mathrm{sVOLE}}^{\mathbb{F}_p,\mathbb{F}_{p^r}}$, which returns commitments on random values, denoted by $[\boldsymbol{\nu}]$, where $\boldsymbol{\nu} \in \mathbb{F}_p^n$.
3. $\mathcal{P}$ and $\mathcal{V}$ send (`Extend`, $\sum_{i=1}^{d}(7k_i+1)+1$) to $\mathcal{F}_{\mathrm{sVOLE}}^{\mathbb{F}_p,\mathbb{F}_{p^r}}$, which returns commitments on random values, denoted by $,[\boldsymbol{\mu}],[\pi]$, where $\boldsymbol{\mu} \in \mathbb{F}_{p^r}^{\sum_{i=1}^{d}(7k_i+1)}$, $\pi \in \mathbb{F}_{p^r}$. For simplicity, we view $\boldsymbol{\mu}$ as $\{\mu_{i,j}\}_{i\in[0,d),j\in[7k_{i+1}+1]}$.

Online phase

1. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ obtain $[\mathbf{w}]$ (by $\mathcal{P}$ sending $\boldsymbol{\delta} := \mathbf{w} - \boldsymbol{\nu}$ to $\mathcal{V}$).
2. For each layer i, $\mathcal{P}$ computes $\mathbf{W}_i$ and stores them. $\mathcal{V}$ sends a random $\boldsymbol{r}_0 \in \mathbb{F}_{p^r}^{k_0}$ to $\mathcal{P}$, then they locally compute $m_0 := \widetilde{W}_0(\boldsymbol{r}_0)$ (Note that $\mathbf{W}_0 = \mathbf{1} \in \mathbb{F}_{p^r}^{s_0}$).
3. For $i = 0, 1, \ldots, d-1$,
   (a) The prover $\mathcal{P}$ defines the $2k_{i+1}$-variate polynomial $f^{(i)}_{\boldsymbol{r}_i}(\mathbf{X},\mathbf{Y}) := \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_i,\mathbf{X},\mathbf{Y})\widetilde{W}_{i+1}(\mathbf{X})\widetilde{W}_{i+1}(\mathbf{Y}) + \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i,\mathbf{X},\mathbf{Y})(\widetilde{W}_{i+1}(\mathbf{X}) + \widetilde{W}_{i+1}(\mathbf{Y}))$.
   (b) For $j = 1, \ldots, k_{i+1}$,
      i. $\mathcal{P}$ computes a univariate polynomial $g^{(i,j)}(X_j)$ of degree-2, writing as $g_0^{(i,j)} + g_1^{(i,j)} \cdot X_j + g_2^{(i,j)} \cdot X_j^2$. $\mathcal{P}$ sends $g_0^{(i,j)} - \mu_{i,3j-2}$, $g_1^{(i,j)} - \mu_{i,3j-1}$, $g_2^{(i,j)} - \mu_{i,3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([g_0^{(i,j)}],[g_1^{(i,j)}],[g_2^{(i,j)}])$, denoted by $[\![g^{(i,j)}(\cdot)]\!]$.
      ii. $\mathcal{V}$ samples $\bar{x}_j^{(i)} \xleftarrow{\$} \mathbb{F}_{p^r}$ and sends it to $\mathcal{P}$.
   (c) For $j = 1, \ldots, k_{i+1}$,
      i. $\mathcal{P}$ computes a single variable polynomial $h^{(i,j)}(Y_j)$ of degree 2, writing as $h_0^{(i,j)} + h_1^{(i,j)} \cdot Y_j + h_2^{(i,j)} \cdot Y_j^2$. $\mathcal{P}$ sends $h_0^{(i,j)} - \mu_{i,3k_{i+1}+3j-2}$, $h_1^{(i,j)} - \mu_{i,3k_{i+1}+3j-1}$, $h_2^{(i,j)} - \mu_{i,3k_{i+1}+3j}$ to $\mathcal{V}$. They essentially obtain a triple of commitments $([h_0^{(i,j)}],[h_1^{(i,j)}],[h_2^{(i,j)}])$, denoted by $[\![h^{(i,j)}(\cdot)]\!]$.
      ii. $\mathcal{V}$ samples $\bar{y}_j^{(i)} \xleftarrow{\$} \mathbb{F}_{p^r}$ and sends it to $\mathcal{P}$.
   (d) Let $L^{(i)}$ be the unique line satisfying $L^{(i)}(0) = \bar{\boldsymbol{x}}^{(i)}$, $L^{(i)}(1) = \bar{\boldsymbol{y}}^{(i)}$. $\mathcal{P}$ computes a univariate polynomial $q^{(i)}(X)$ by restricting $\widetilde{W}_{i+1}$ to $L^{(i)}$, writing as $\sum_{j=0}^{k_{i+1}} q_j^{(i)} \cdot X^j$. $\mathcal{P}$ sends $(q_0^{(i)} - \mu_{i,6k_{i+1}+1}, \ldots, q_{k_{i+1}}^{(i)} - \mu_{i,7k_{i+1}+1})$ to $\mathcal{V}$, and similarly, they obtain $([q_0^{(i)}],\ldots,[q_{k_{i+1}}^{(i)}])$, denoted by $[\![q^{(i)}(\cdot)]\!]$.
   (e) $\mathcal{V}$ selects $r^{(i)} \xleftarrow{\$} \mathbb{F}_{p^r}$ and sends it to $\mathcal{P}$. $\mathcal{P}$ computes $m_{i+1} := q^{(i)}(r^{(i)})$. Then they set $\boldsymbol{r}_{i+1} := L^{(i)}(r^{(i)}) \in \mathbb{F}_{p^r}^{k_{i+1}}$, and compute $[m_{i+1}] := [q^{(i)}(r^{(i)})]$.
4. $\mathcal{P}$ and $\mathcal{V}$ perform the following checks.
   (a) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Lin` in Figure 2 on input tuples $\{([m_i],[\![g^{(i,j)}(\cdot)]\!],[\![h^{(i,j)}(\cdot)]\!])\}_{i\in[0,d),j\in[1,k_{i+1}]}$.
   (b) $\mathcal{P}$ and $\mathcal{V}$ run the procedure `Batch-Mult` in Figure 2 on input tuples $\{([\![q^{(i)}(\cdot)]\!],[\![h^{(i,k_{i+1})}(\cdot)]\!])\}_{i\in[0,d)}$.
   (c) $\mathcal{P}$ opens $[m_d] - \sum_{\omega\in\{0,1\}^{k_d}}[W_d(\omega)] \cdot \boldsymbol{\chi}_\omega(\boldsymbol{r}_d)$, where $\boldsymbol{\chi}_\omega(\cdot)$ is a Lagrange basis as defined in Def. 1. $\mathcal{V}$ checks whether it is a valid opening of $[0]$.
5. $\mathcal{V}$ accepts if and only $\mathcal{P}$ passes all the checks above. Otherwise, $\mathcal{V}$ rejects.

Fig. 12: Our ZK protocol for layered circuits over $\mathbb{F}_p$ in the $\mathcal{F}_{\mathrm{sVOLE}}^{\mathbb{F}_p,\mathbb{F}_{p^r}}$-hybrid model.