Doubly-efficient zkSNARKs without trusted setup

Riad S. Wahby⋆ rsw@cs.stanford.edu
Ioanna Tzialla° iontzialla@gmail.com
abhi shelat† abhi@neu.edu
Justin Thaler‡ justin.thaler@georgetown.edu
Michael Walfish° mwalfish@cs.nyu.edu

⋆Stanford °NYU †Northeastern ‡Georgetown

Abstract. We present a zero-knowledge argument for NP with low communication complexity, low concrete cost for both the prover and the verifier, and no trusted setup, based on standard cryptographic assumptions (DDH). Specifically, communication is proportional to the square root of the size of the witness, plus $d \cdot \log(G)$ where d is the depth and G is the width of the verifying circuit. When applied to batched or data-parallel statements, the prover's cost is linear and the verifier's cost is sub-linear in the verifying circuit size, both with good constants. Together, these properties represent a new point in the tradeoffs among setup, complexity assumptions, proof size, and computational cost.

Our argument is public coin, so we apply the Fiat-Shamir heuristic to produce a zero-knowledge succinct non-interactive argument of knowledge (zkSNARK), which we call Hyrax. We evaluate Hyrax on three benchmarks, SHA-256 Merkle trees, image transformation, and matrix multiplication. We find that Hyrax scales to 6–27× larger circuit sizes than a highly-optimized prior system, and that its proofs are 2–10× smaller than prior work with similar properties.

1 Introduction

A zero-knowledge proof is a proof that convinces a verifier while revealing nothing but its own validity. Since they were introduced by Goldwasser, Micali, and Rackoff [52], zero-knowledge (ZK) proofs have found applications in domains as diverse as authentication and signature schemes [81, 84], secure encryption [41, 83], and emerging blockchain technologies [19].

A seminal result in the theory of interactive proofs and cryptography is that any problem solvable by an efficient interactive proof (IP) is also solvable by an efficient (computational) zero-knowledge proof or (perfect) zero-knowledge argument [15]. Unfortunately, existing transformations from interactive proofs to zero-knowledge proofs or arguments lead to large concrete or asymptotic blowups in cost.

For example, early techniques for designing zero-knowledge proofs reduce a given problem to an NP-complete problem such as 3-coloring [50] or Hamiltonian path [26]. Such reductions incur polynomial blowups in the work and communication involved in the proof, and these techniques further require many repetitions to achieve soundness. (More recently, this line of work has been improved to build succinct ZK arguments for NP statements using algebraic commitments [27, 29, 30, 53, 54, 58], but these constructions generally entail many expensive cryptographic operations.) Other early works avoid these large overheads, but are tailored to specific problems with algebraic structure and cryptographic significance, most notably Schnorr-style proofs [84] for languages related to statements about the discrete logarithms of group elements.

Several recent lines of work have sought to remain fully general while avoiding the large overheads implicit in earlier results. Unfortunately, as detailed in Section 2, these works still yield costly protocols or come with significant limitations. In particular, state-of-the-art, general-purpose ZK protocols suffer from one or more of the following problems: (a) they require proof size that is linear or super-linear in the size of the computation verifying an NP witness; (b) they require the prover or verifier to perform work that is super-linear in the time to verify a witness; (c) they require a complex parameter setup to be performed by a trusted party; or (d) they rely on non-standard cryptographic assumptions. These issues have limited the use of such general-purpose ZK proof systems in many contexts.

Our goal in this work is to address the limitations of existing general-purpose zero-knowledge proofs and arguments. Specifically, we would like to take any computation for verifying an NP statement and turn it into a zero-knowledge proof of that statement's validity. Our desiderata are that

- the proof should be succinct (i.e., sub-linear in the size of the statement and the witness to the statement's validity);
- the verifier should run in time linear in the input size plus the proof size;
- the prover, given a witness to the statement's validity, should run in time linear in the cost of the NP verification procedure;
- the scheme should not require a setup phase or a trusted common reference string; and
- soundness and zero knowledge should be statistical or based on standard cryptographic assumptions.

We note, that although our implementation and evaluation occurs in the random oracle model [14], our protocols can be instantiated in the plain (interactive, honest-verifier) model.

Techniques. Ben-Or et al. [15] and Cramer and Damgård [39] show how to transform IPs into (computationally) ZK proofs or (perfectly) ZK arguments using cryptographic commitment schemes. At a high level, rather than sending its messages "in the clear", the IP prover sends cryptographic *commitments* corresponding to its messages. These commitments are *binding*, ensuring that the prover cannot equivocate about its messages. They are also *hiding*, meaning that the verifier cannot learn the committed value (thus ensuring zero knowledge). Finally, the commitment scheme has a homomorphism property (§3.1) that allows the verifier to check the prover's messages "underneath the commitments," ensuring that the resulting proof or argument is secure against cheating provers—even though committed

values are never revealed.

Accepted wisdom is that such transformations introduce large overheads (e.g., [34, Section 1.1]). In this paper, we challenge that wisdom by constructing *Hyrax*, a zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) that is concretely efficient and meets all of our desiderata for many cases of interest. Hyrax is efficient for four reasons: first, it builds on a state-of-the-art interactive proof, with a slight optimization to reduce communication complexity (§3.2). Second, it tightly integrates this IP with a commitment scheme, reducing computational cost by 4–5× for both prover and verifier (§5). Third, it exploits the structure of the IP to adapt an efficient ZK proof for special algebraic languages, resulting in an asymptotic reduction in both the prover's communication and the verifier's computation (§6). Finally, it uses cryptographic operations (required by the commitment scheme) only for the witness and for the IP prover's messages, which are sub-linear in the size of the arithmetic circuit that verifies the witness. In contrast, many recent works (§2) invoke cryptographic primitives for *each gate* in the verifying circuit [19, 20, 23, 27, 47, 78].

Contributions. All told, we design, implement and evaluate a public-coin, succinct, perfect zero-knowledge argument of knowledge for arithmetic circuit (AC) satisfiability (§3.1). Our argument system has the following properties when applied to a witness w and an AC C having width G and depth d:

- It is built on standard hardness assumptions (e.g., DDH) and requires no trusted setup.
- Its proofs are *sub-linear* in $|C|$, requiring only $10d \log G + 2\sqrt{|w|}$ group elements.
- Its prover runs in time *linear* in $|C|$, with good constants, if C has sufficient parallelism (practically, a few tens of parallel instances suffices) and requires only $O(d \log G + |w|)$ cryptographic operations (again, with good constants).
- Its verifier runs in time *sub-linear* in $|C|$, if C has sufficient parallelism.[1] Specifically, it runs in time proportional to $|x|$ plus the proof size, with good constants.

Since it is public coin, we can compile it into a zkSNARK in the random oracle model using the Fiat-Shamir heuristic [42]. We evaluate this zkSNARK on three benchmarks, SHA-256 Merkle trees, image resizing, and matrix multiplication (§8). We find that it scales to 6–27× larger ACs than the highly-optimized libsnark [9, 23] and that its proofs are 2–10× smaller than prior work with similar properties. And although our current implementation (§7) is in Python, and thus slow, careful cost modeling (§8.2) suggests that optimization will yield prover and verifier runtimes competitive with or better than prior work.

2 Related work

Kilian [65, 66] and later Micali [73] show how to construct succinct arguments for NP from probabilistically checkable proofs (PCPs). Recent work on PCPs has improved upon these techniques, and the latest approaches extend the classical notion of PCPs by adding interaction; these are interactive oracle proofs (IOPs) [17, 21], also known as probabilistically checkable interactive proofs [82]. However, all of these works incur many-orders-of-magnitude overheads for the prover in runtime and memory usage, and published implementations [17] are not zero knowledge. A recent pre-print by Ben-Sasson et al. [18] improves the concrete efficiency of IOPs; in related unpublished work, the authors present evidence[2] that their implementation efforts are promising for some classes of problems, but the resulting prover still appears to be highly memory-intensive.

Ishai, Kushilevitz, and Ostrovsky [61] observe that a *linear* PCP can be combined with an additively homomorphic encryption scheme in order to construct a more efficient argument, albeit in the pre-processing model; several refinements and implementations exist [31, 85–87]. Groth [55] and later Lipmaa [69] make this idea non-interactive. Gennaro, Gentry, Parno, and Raykova [47] present a very efficient linear PCP[3] that forms the basis of many recent zkSNARK implementations [12, 13, 17, 19, 20, 22, 23, 31, 35, 37, 40, 43–45, 67, 75, 78, 98]. The latest instantiation of this paradigm by Groth [56] has a proof comprising 3 group elements that requires only 3 elliptic-curve pairing operations to verify. However, all of these zkSNARKs rely on non-standard, non-falsifiable knowledge of exponent assumptions or work in the generic group model [89]. Moreover, they require a trusted party to create a (structured) common reference string that is as large as the verifying circuit. Finally, although these systems have been deployed in popular applications such as ZCash [8, 19], the prover's runtime is quasi-linear in the circuit size and includes a few public key operations for each gate in the circuit; and in practice, memory overheads limit the instances sizes that these systems can handle [98].

Another approach due to Ishai, Kushilevitz, Ostrovsky, and Sahai [62] transforms a secure multi-party computation protocol into a zero-knowledge argument. This approach gives zero-knowledge arguments for constant-depth circuits with proof size polylogarithmic in the circuit size and linear in the witness size; for general circuits, proof size is linear in the circuit size. Giacomelli, Madsen, and Orlandi apply this approach to an efficient 3-party secure protocol to construct ZKBoo [48], a ZK argument system for Boolean circuits. ZKB++, a follow-up improvement by Chase et al. [32], reduces proof size by constant factors. These schemes are efficient for small circuits, but their communication and computation costs are linear in circuit size.

Ames et al. [10] improve the IKOS transformation and apply it to a more sophisticated secure computation protocol. Their scheme, Ligero, is a public-coin, zero-knowledge argument system without trusted setup that only requires collision-resistant hash functions (and no public-key cryptography). Ligero proves the satisfiability of a circuit C with communication complexity $\tilde{O}(\sqrt{|C|})$, but both the prover and the verifier perform work quasi-linear in $|C|$. Our approach gives smaller proofs for languages with small witnesses, but requires public-key cryptography.

Bootle et al. [27] give two ZK arguments for AC satisfiability from hardness of discrete logarithms, building on ideas of

[1] Even without data parallelism, the verifier runs in time sub-linear in $|C|$ if C's wiring pattern satisfies a technical "regularity" condition [36, 51].

[2] In a talk, "Concretely efficient Computational Integrity (CI) from PCPs" [16].

[3] The observation that quadratic span programs can be viewed as linear PCPs is due to Bitansky et al. [25] and Setty et al. [85].

Groth [54]. The first argument has communication proportional to $\sqrt{M}$ for an AC with M multiplications. The second reduces this to log M at the cost of significantly higher concrete costs for the prover and verifier. A key difference from our approach is that the verifier in both of these schemes has work proportional to M, while our verifier is sub-linear in many cases. Very recently, Bootle et al. [28] extend these ideas further to give a ZK argument with proof size $O(\sqrt{C})$ whose verifier uses $O(C)$ *additions* (which are concretely less expensive than multiplications). The authors state that the hidden constants are large and do not recommend implementing the system as-is.

Most similar to our work, Zhang et al. [99] show how to use an interactive proof [36, 51, 90] along with a verifiable polynomial delegation scheme [64, 77] to construct a succinct interactive argument. Although that work does not address zero knowledge, a follow-up [100] (concurrent with and independent from our work) achieves ZK using the same commit-and-prove technique that we use, with several key differences. First, their commitment to the witness w requires a trusted setup to create public parameters (of size $O(|w|)$) and relies on non-standard, non-falsifiable assumptions; on the other hand, this approach reduces $\mathcal{P}$'s communication to logarithmic in $|w|$. Second, their argument system is based on a different IP than ours; theirs requires slightly more communication. Finally, our method of compiling the IP into a ZK argument uses additional refinements (§5) that reduce costs. Both our IP and our refinements apply to their work; we estimate that they would reduce $\mathcal{P}$'s communication by ≈3× and $\mathcal{V}$'s computational costs by ≈5×.

3 Background

3.1 Definitions

We use $\langle A(z_a), B(z_b)\rangle(x)$ to denote the random variable representing the (local) output of machine B when interacting with machine A on common input x, when the random tapes for each machine are uniformly and independently chosen, and A and B has auxiliary inputs z_a and z_b respectively. We use $\mathrm{tr}\langle A(z_a), B(z_b)\rangle(x)$ to denote the random variable representing the entire transcript of the interaction between A and B, and $\mathsf{View}(\langle A(z_a), B(z_b)\rangle(x))$ to denote the distribution of the transcript. The symbol $\approx_c$ denotes that two ensembles are computationally indistinguishable.

Arithmetic circuits

Sections 3.2 and 3.2 focus on the arithmetic circuit (AC) *evaluation* problem. In this problem, one fixes an arithmetic circuit C, consisting of addition and multiplication gates over a finite field $\mathbb{F}$. We assume throughout that C is layered, with all gates having fan-in at most 2 (any arithmetic circuit can be made layered while increasing the number of gates by a factor of at most the circuit depth). C has depth d and input x with length $|x|$. The goal is to evaluate C on input x. In an interactive proof or argument for this problem, the prover sends the claimed outputs y of C on input x, and must prove that $y = C(x)$.

Our end goal in this work is to give efficient protocols for the arithmetic circuit *satisfiability* problem. Let $C(\cdot,\cdot)$ be a layered arithmetic circuit of fan-in two. Given an input x and outputs y, the goal is to determine whether there exists a *witness* w such that $C(x, w) = y$. The corresponding witness relation for this problem is the natural one: $R_{(x,y)} = \{w\colon C(x, w) = y\}$.

Interactive protocols and zero knowledge

Definition 1 (Interactive arguments and proofs)**.** *A pair of probabilistic interactive machines $\langle\mathcal{P},\mathcal{V}\rangle$ is called an interactive argument system for a language L if there exists a negligible function η such that the following two conditions hold:*

1. *Completeness: For every $x \in L$ there exists a string w s.t. for every $z \in \{0,1\}^*$, $\Pr[\langle\mathcal{P}(w),\mathcal{V}(z)\rangle(x)=1] \geq 1-\eta(|x|)$.*
2. *Soundness: For every $x \notin L$, every interactive PPT $\mathcal{P}^*$, and every $w, z \in \{0,1\}^*$, $\Pr[\langle\mathcal{P}^*(w),\mathcal{V}(z)\rangle(x)=1] \leq \eta(|x|)$.*

If soundness holds against computationally unbounded cheating provers $\mathcal{P}^*$, then $\langle\mathcal{P},\mathcal{V}\rangle$ is called an interactive proof (IP).

Definition 2 (Zero knowledge (ZK))**.** *Let $L \subset \{0,1\}^*$ be a language and for each $x \in L$, let $R_x \subset \{0,1\}^*$ denote a corresponding set of* witnesses *for the fact that $x \in L$. Let R_L denote the corresponding language of valid (input, witness) pairs, i.e., $R_L = \{(x,w)\colon x \in L \text{ and } w \in R_x\}$. An interactive proof or argument system $\langle\mathcal{P},\mathcal{V}\rangle$ for L is computational zero-knowledge (CZK) with respect to an auxiliary input if for every PPT interactive machine $\mathcal{V}^*$, there exists a PPT algorithm S, called the* simulator*, running in time polynomial in the length of its first input, such that for every $x \in L$, $w \in R_x$, and $z \in \{0,1\}^*$,*

$$\mathsf{View}(\langle P(w),\mathcal{V}^*(z)\rangle(x)) \approx_c S(x,z) \tag{1}$$

when the distinguishing gap is considered as a function of $|x|$. If the statistical distance *between the two distributions is negligible, then the interactive proof or argument system is said to be* statistical *zero-knowledge (SZK). If the simulator is allowed to abort with probability at most $1/2$, but the distribution of its output conditioned on not aborting is* identically *distributed to $\mathsf{View}(\langle P(w),\mathcal{V}^*(z)\rangle(x))$, then the interactive proof or argument system is called* perfect *zero-knowledge (PZK).*

The left term in Equation (1) denotes the view of $\mathcal{V}^*$ after it interacts with $\mathcal{P}$ on common input x; the right term denotes the distribution of the output of the simulator S on x. For any CZK (resp., SZK or PZK) protocol, Def. 2 requires the simulator to produce a distribution that is computationally (respectively, statistically or perfectly) indistinguishable from the verifier's view in the zero-knowledge proof or argument system.

Our zero-knowledge arguments also satisfy a proof of knowledge (PoK) property. Intuitively, this means that in order to produce a convincing proof of a statement, the prover must *know* a witness to the validity of the statement. To define this notion formally, we follow Groth and Ishai [57] who borrow the notion of statistical witness-extended emulation from Lindell [68]. This notion of PoK suffices for most cryptographic applications.

Definition 3 (Witness-extended emulation)**.** *Let L be a language and R_L corresponding language of valid (input, witness) pairs as in Definition 2. An interactive argument system $\langle\mathcal{P},\mathcal{V}\rangle$ for L has* witness-extended emulation *if for all deterministic polynomial time $\mathcal{P}^*$ there exists an expected polynomial time emulator E such that for all non-uniform polynomial time adversaries A, the following two probabilities differ by at most a negligible*

function in the security parameter λ:

$$\Pr\left[(u,s) \leftarrow A(1^\lambda); t \leftarrow \mathsf{tr}\langle \mathcal{P}^*(u,s), \mathcal{V}(u)\rangle : A(t) = 1\right]$$

$$\textit{and } \Pr\left[\begin{array}{l}(u,s) \leftarrow A(1^\lambda); (t,w) \leftarrow E^{\mathcal{P}^*(u,s)}(u) : A(t) = 1 \wedge \\ \textit{if } t \textit{ is an accepting transcript, then } R_L(u,w) = 1.\end{array}\right]$$

Here, the oracle called by E permits rewinding the prover to a specific point and resuming with fresh randomness for the verifier from this point onwards.

The sub-protocols we develop in the refinements of Sections 5 and 6 are *generalized special sound*, which implies witness-extended emulation (Appx. A.4).

Definition 4 (Generalized special soundness)**.** *A $(2\mu+1)$-move interactive argument $\langle \mathcal{P}, \mathcal{V}\rangle$ is* generalized special sound *if there exists a PPT algorithm E that extracts a witness except with negligible probability given an $(n_1, \ldots, n_\mu)$-tree of accepting transcripts, This tree comprises n_1 transcripts with fresh randomness in $\mathcal{V}$'s first message; and for each such transcript, n_2 transcripts with fresh randomness in $\mathcal{V}$'s second message; etc., for a total of $\prod_1^\mu n_i$ leaves.*

Note that (standard) special soundness is $\mu = 1, n_1 = 2$.

Commitment schemes

Informally, a commitment scheme allows a *sender* to produce a message $C = \mathsf{Com}(m)$ that hides m from a *receiver* but binds the sender to the value m. In particular, when the sender *opens* C and reveals m, the receiver is convinced that this was indeed the sender's original value.

Definition 5 (Collection of non-interactive commitments [60])**.** *We say that a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Com})$ *is a* collection of non-interactive commitments *if the following conditions hold:*

- **Computational binding:** *For every (non-uniform) PPT A, there is a negligible function η such that for every $n \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} i \leftarrow \mathsf{Gen}(1^n)\ ; \\ (m_0, r_0), (m_1, r_1) \leftarrow A(1^n, i)\ : \\ m_0 \neq m_1, |m_0| = |m_1| = n, \\ \mathsf{Com}_i(m_0; r_0) = \mathsf{Com}_i(m_1; r_1)\end{array}\right] \leq \eta(n)$$

- **Perfect hiding:** *For $i \in \{0,1\}^*$, and for any two $m_0, m_1 \in \{0,1\}^*$ where $|m_0| = |m_1|$, the ensembles $\{\mathsf{Com}_i(m_0)\}_{n\in\mathbb{N}}$ and $\{\mathsf{Com}_i(m_1)\}_{n\in\mathbb{N}}$ are identically distributed.*

Remark 1. *We define only the computational variant of binding and the perfect variant of hiding because the commitment schemes used in our implementation satisfy these properties. The use of such commitment schemes in our context yields a PZK argument. If we instead used perfectly (or statistically) binding, computationally hiding commitments, we would obtain a CZK proof.*

We say that $\mathsf{Com}(m; r)$ is a commitment to m with *opening* r; the sender chooses r at random. To open the commitment, the sender reveals (m, r). We sometimes leave the opening value implicit, e.g., $\mathsf{Com}(m)$.

Collections of non-interactive commitments can be constructed based on any one-way function [59, 74], but we require a *homomorphism* property (defined below) that these commitments do not provide. (The Pedersen commitment [79], described in Appx. A, provides this property.)

Definition 6 (Additive homomorphism)**.** *Given* $\mathsf{Com}(x; s_x)$ *and* $\mathsf{Com}(y; s_y)$*, it holds that*

$$\mathsf{Com}(x; s_x) \odot \mathsf{Com}(y; s_y) = \mathsf{Com}(x + y; s_x + s_y)$$

In a vector *or* multi-*commitment scheme (i.e., if x and y are vectors), the additive homomorphism is vector-wise.*

3.2 Our starting point: Gir^{++} (Giraffe, with a tweak)

The most efficient known IPs for the AC evaluation problem (§3.1) follow a line of work starting with the breakthrough result of Goldwasser, Kalai, and Rothblum (GKR) [51]. Cormode, Mitzenmacher, and Thaler (CMT) [36] and Vu et al. [95] refine this result, giving $O(|C| \log |C|)$ prover and $O(|x| + d \log |C|)$ verifier run times, for an AC C with depth d and input x.

Further refinements are possible in the case where C is *data parallel*, meaning it consists of N identical sub-computations run on different inputs. (We refer to each sub-computation as a *sub-AC* of C, and we assume for simplicity that all layers of the sub-AC have width G, so $|C| = d \cdot N \cdot G$.) Thaler [90] reduced the prover's runtime in the data-parallel case from $O(|C| \log |C|)$ to $O(|C| \log G)$. Very recently, Wahby et al. introduced Giraffe [97], which reduces the prover's runtime to $O(|C| + d \cdot G \cdot \log G)$. Since $|C| = d \cdot N \cdot G$, observe that when $N \geq \log G$, the time reduces to $O(|C|)$, which is asymptotically optimal. That is, whenever there is a sufficiently large amount of data parallelism, the prover's runtime is just a constant factor slower than evaluating the circuit gate-by-gate without providing any proof of correctness.

Our work builds on *Gir*$^{++}$, a version of Giraffe that incorporates a small modification to reduce the prover's communication cost. In our description of Gir^{++}, we use the following notation. Assume for simplicity that N and G are powers of 2, and let $b_N = \log_2 N$ and $b_G = \log_2 G$. Within a layer of C, each gate is labeled with a pair $(i, j) \in \{0,1\}^{b_N} \times \{0,1\}^{b_G}$. Number the layers of C from 0 to d in reverse execution order, so that 0 refers to the output layer, and d refers to the input layer. Each layer i is associated with an evaluator function $V_i : \{0,1\}^{b_N} \times \{0,1\}^{b_G} \to \mathbb{F}$ that maps a gate's label to the output of that gate when C is evaluated on input x. For example, $V_0(i, j)$ is the j'th output of the i'th sub-AC, and $V_d(i, j)$ is the jth input to the ith sub-AC.

At a high level, the protocol proceeds in iterations, one for each layer of the circuit. At the start of the protocol, the prover $\mathcal{P}$ sends the claimed outputs y of C (i.e., all the claimed evaluations of V_0). The first iteration of the protocol reduces the claim about V_0 to a claim about V_1, in the sense that it is safe for the verifier $\mathcal{V}$ to believe the former claim as long as $\mathcal{V}$ is convinced of the latter. But $\mathcal{V}$ cannot directly check the claim about V_1, because doing so would require evaluating all of the gates in C other than the outputs themselves. Instead, the second iteration reduces the claim about V_1 to a claim about V_2, and so on, until $\mathcal{P}$ makes a claim about V_d (i.e., the inputs to C), which $\mathcal{V}$ can check itself.

To describe how the reduction from a claim about V_i to a claim about V_{i+1} is performed, we first introduce *multilinear extensions*, *the sum-check protocol*, and *wiring predicates*.

Multilinear extensions. An *extension* of a function $f\colon \{0,1\}^\ell \to \mathbb{F}$ is a ℓ-variate polynomial g over $\mathbb{F}$ such that $g(x) = f(x)$ for all $x \in \{0,1\}^\ell$. Any such function f has a *unique* multilinear extension (MLE), denoted $\tilde{f}$ (i.e., $\tilde{f}$ is the unique multilinear polynomial such that $\tilde{f}(x) = f(x)$ for all $x \in \{0,1\}^\ell$). Given a vector $z \in \mathbb{F}^m$ with $m = 2^\ell$, we will often view z as a function $V_z\colon \{0,1\}^\ell \to \mathbb{F}$ mapping indices to vector entries, and use $\tilde{V}_z$ to denote the MLE of V_z.

The sum-check protocol. Fix an ℓ-variate polynomial g over $\mathbb{F}$, and let $\deg_i(g)$ denote the degree of g in variable i. The sum-check protocol [70] is an interactive proof that allows $\mathcal{P}$ to convince $\mathcal{V}$ that a claim about the value of $\sum_{x\in\{0,1\}^\ell} g(x)$ can be reduced to a claim about the value of $g(r)$ where $r \in \mathbb{F}^\ell$ is a point randomly chosen by $\mathcal{V}$. There are ℓ rounds, and $\mathcal{V}$'s runtime is $O(\sum_{i=1}^{\ell} \deg_i(g))$ plus the cost of evaluating $g(r)$. The mechanics of the sum-check protocol (i.e., $\mathcal{P}$'s prescribed messages and $\mathcal{V}$'s checks) are described in more detail in Section 4.

Wiring predicates capture the wiring information of the sub-ACs. Define the wiring predicate $\text{add}_i\colon \{0,1\}^{3b_G} \to \{0,1\}$, where $\text{add}_i(g, h_0, h_1)$ returns 1 if (a) within each sub-AC, gate g at layer $i-1$ is an add gate and (b) the left and right inputs of g are, respectively, h_0 and h_1 at layer i (and 0 otherwise). mult_i is defined analogously for multiplication gates. We also define the equality predicate $\text{eq} : \{0,1\}^{2b_N} \to \{0,1\}$ by $\text{eq}(a,b) = 1$ if and only if $a = b$.

Thaler [90] showed how to express $\tilde{V}_{i-1}$ in terms of $\tilde{V}_i$: for $(q', q) \in \mathbb{F}^{b_N} \times \mathbb{F}^{b_G}$, let $P_{q',q,i}\colon \mathbb{F}^{\log N} \times \mathbb{F}^{\log G} \times \mathbb{F}^{\log G} \to \mathbb{F}$ denote the following polynomial:

$$\begin{aligned} P_{q',q,i}(r', r_0, r_1) = \\ \tilde{\text{eq}}(q', r') \cdot \big[\tilde{\text{add}}_i(q, r_0, r_1)\left(\tilde{V}_i(r', r_0) + \tilde{V}_i(r', r_1)\right) + \\ \tilde{\text{mult}}_i(q, r_0, r_1)\left(\tilde{V}_i(r', r_0) \cdot \tilde{V}_i(r', r_1)\right)\big] \end{aligned}$$

Then we have

$$\tilde{V}_{i-1}(q', q) = \sum_{h_0, h_1 \in \{0,1\}^{b_G}} \sum_{h' \in \{0,1\}^{b_N}} P_{q',q,i}(h', h_0, h_1). \quad (2)$$

Protocol overview

Step 1. At the start of the protocol, $\mathcal{P}$ sends the claimed output y, thereby specifying a function $V_y\colon \{0,1\}^{b_G+b_N} \to \mathbb{F}$ mapping the label of each output gate to the corresponding entry of y. The verifier needs to check that $V_y = V_0$ (i.e., that the claimed outputs equal the correct outputs of C on input x). To accomplish this, it is enough to check that $\tilde{V}_y = \tilde{V}_0$, which $\mathcal{V}$ does by choosing a random pair $(q', q) \in \mathbb{F}^{b_N} \times \mathbb{F}^{b_G}$ and checking that $\tilde{V}_y(q', q) = \tilde{V}_0(q', q)$. If this check passes, then the Schwartz-Zippel lemma implies that it is safe for $\mathcal{V}$ to believe that $\tilde{V}_y = \tilde{V}_0$. The verifier can compute $\tilde{V}_y(q', q)$ on its own in $O(NG)$ time, but cannot do the same for $\tilde{V}_0(q', q)$ since this would require $\mathcal{V}$ to evaluate C. Fortunately, Equation (2) expresses $\tilde{V}_{i-1}(q', q)$ in a form that is amenable to outsourcing via the sum-check protocol.

Step 2. $\mathcal{P}$ and $\mathcal{V}$ apply the sum-check protocol to the polynomial $P_{q',q,1}$. At the end of the sum-check protocol, $\mathcal{V}$ must evaluate $P_{q',q,1}$ at a random input (r', r_0, r_1), which requires the values $\tilde{V}_1(r', r_0)$ and $\tilde{V}_1(r', r_1)$.

Rather than evaluate these directly (which would be too costly), $\mathcal{V}$ asks $\mathcal{P}$ to send (purported) evaluations of $\tilde{V}_1$ at these points, which $\mathcal{V}$ uses to evaluate $P_{q',q,1}$. $\mathcal{V}$ then reduces $\mathcal{P}$'s claim about $\tilde{V}_1$ to an expression, in terms of $\tilde{V}_2$, that can be checked with another invocation of the sum-check protocol.

Gir^{++} handles this differently than prior work [51, 90, 97]; we detail both approaches below.

Final steps. The protocol's second iteration repeats step 2, allowing $\mathcal{V}$ to check $\mathcal{P}$'s claim about $\tilde{V}_1$ using $\mathcal{V}$'s expression in terms of $\tilde{V}_2$; the result is a claim about $\tilde{V}_2$ and a corresponding expression, in terms of $\tilde{V}_3$, that can once again be checked with a sum-check invocation. This continues, layer by layer, until $\mathcal{V}$ obtains a claim about $\tilde{V}_d$. $\mathcal{V}$ can check this assertion because $\tilde{V}_d = \tilde{V}_x$, which $\mathcal{V}$ can evaluate directly.

Reducing from $\tilde{V}_i$ to $\tilde{V}_{i+1}$

Gir^{++} differs from Giraffe only in that they use different techniques, described below, to reduce $\mathcal{P}$'s claim at the end of step 2 into the expression that $\mathcal{V}$ and $\mathcal{P}$ use for the next sum-check invocation. In particular, Giraffe's $\mathcal{V}$ uses the "reducing from two points to one point" technique at the end of step 2, whereas Gir^{++} uses the "random linear combination" technique.[4] An exception is after the final sum-check invocation, when Gir^{++} uses the same approach as Giraffe; we discuss below.

This optimization reduces $\mathcal{P}$'s communication cost in Gir^{++} compared to Giraffe by up to 33%, depending on the amount of data parallelism. While this would also reduce costs for Giraffe, reducing $\mathcal{P}$'s communication has even greater importance in our setting. This is because, after compiling this protocol into a zero-knowledge argument (§4), each field element sent by $\mathcal{P}$ in the underlying IP increases the proof size and entails expensive cryptography. Hence, reduces $\mathcal{P}$'s communication in Gir^{++} reduces the cost of our zero-knowledge argument.

Reducing from two points to one point. Recall that at the end of the sum-check in step 2, $\mathcal{V}$ must evaluate $\tilde{V}_i$ at (r', r_0) and (r', r_1) in order to evaluate $P_{q',q,i}$. Prior work [51, 90, 97] accomplishes this by requiring $\mathcal{P}$ to send $\mathcal{V}$ the restriction of $\tilde{V}_i$ to the the unique line H in $\mathbb{F}^{b_N+b_G}$ passing through the points (r', r_0) and (r', r_1). This is a univariate polynomial of degree at most b_G that implicitly contains claims about the value of $\tilde{V}_i$ at all points on H. $\mathcal{V}$ should believe these claims as long as $\mathcal{V}$ evaluates the purported restriction of $\tilde{V}_i$ to H at *one* random point (r', r) and verifies that the result equals $\tilde{V}_i(r', r)$; by Equation (2), $\mathcal{V}$ can do this by engaging $\mathcal{P}$ in a sum-check protocol over $P_{r',r,i+1}$. Since the restriction of $\tilde{V}_i$ to H has degree $b_G = \log G$, this approach requires $\mathcal{P}$ to send $\log G + 1$ field elements.

Random linear combination. To reduce $\mathcal{P}$'s communication, Gir^{++}'s verifier instead reduces two claimed evaluations of $\tilde{V}_i$ to one claim about a random linear combination of the two evaluations. To do this, $\mathcal{V}$ requires $\mathcal{P}$ to send two claimed evaluations of $\tilde{V}_i$, v_0 and v_1. $\mathcal{V}$ then samples two field elements

[4] This technique is described in prior theoretical work of Chiesa, Forbes, and Spooner [34], who use it to simplify the analysis of a perfect zero-knowledge protocol in a model known as Interactive PCPs [63].

μ_0, μ_1, and sends them to $\mathcal{P}$. The Schwartz-Zippel lemma implies that it is safe for $\mathcal{V}$ to believe that $\tilde{V}_i(q'_i, q_{i,0}) = v_0$ and $\tilde{V}_i(q'_i, q_{i,1}) = v_1$ if $\mu_0\tilde{V}_i(q'_i, q_{i,0}) + \mu_1\tilde{V}_i(q'_i, q_{i,1}) = \mu_0 v_0 + \mu_1 v_1$. $\mathcal{V}$ can evaluate the LHS by exploiting the fact that

$$\begin{aligned}
&\mu_0\tilde{V}_i(q'_i, q_{i,0}) + \mu_1\tilde{V}_i(q'_i, q_{i,1}) \\
&\quad= \sum_{h_0,h_1\in\{0,1\}^{b_G}} \sum_{h'\in\{0,1\}^{b_N}} \big(\mu_0 \cdot P_{q'_i,q_{i,0},i+1}(h', h_0, h_1) + \\
&\qquad\qquad \mu_1 \cdot P_{q'_i,q_{i,1},i+1}(h', h_0, h_1)\big) \\
&\quad= \sum_{h_0,h_1\in\{0,1\}^{b_G}} \sum_{h'\in b_N} Q_{q'_i,q_{i,0},q_{i,1},\mu_0,\mu_1,i+1}(h', h_0, h_1) \quad (3)
\end{aligned}$$

where $Q_{q',q_0,q_1,\mu_0,\mu_1,i} : \mathbb{F}^{\log N} \times \mathbb{F}^{\log G} \times \mathbb{F}^{\log G} \to \mathbb{F}$ is given by:

$$\begin{aligned}
Q_{q',q_0,q_1,\mu_0,\mu_1,i}(r', r_0, r_1) \triangleq\ & \widetilde{\mathrm{eq}}(q', r') \cdot \\
& \big[\ \big(\mu_0 \cdot \tilde{\mathrm{add}}_i(q_0, r_0, r_1) + \mu_1 \cdot \tilde{\mathrm{add}}_i(q_1, r_0, r_1)\big) \cdot \\
& \qquad \big(\tilde{V}_i(r', r_0) + \tilde{V}_i(r', r_1)\big) \\
& \quad + \big(\mu_0 \cdot \tilde{\mathrm{mult}}_i(q_0, r_0, r_1) + \mu_1 \cdot \tilde{\mathrm{mult}}_i(q_1, r_0, r_1)\big) \cdot \\
& \qquad \big(\tilde{V}_i(r', r_0) \cdot \tilde{V}_i(r', r_1)\big)\ \big]
\end{aligned}$$

This means that $\mathcal{V}$ can compute $\mu_0\tilde{V}_i(q'_i, q_{i,0}) + \mu_1\tilde{V}_i(q'_i, q_{i,1})$ by applying the sum-check protocol to $Q_{q'_i,q_{i,0},q_{i,1},\mu_0,\mu_1,i+1}$. At the end of that sum-check, $\mathcal{V}$ is left with two claims about $\tilde{V}_{i+1}$, to which it applies another sum-check, this time over $Q_{\ldots,i+2}$.

The final sum-check. To avoid increasing $\mathcal{V}$'s computational costs compared to Giraffe, Gir^{++} uses the "reducing from two points to one point" technique after the final sum-check (i.e., the one over $Q_{\ldots,d-1}$). Recall that $\mathcal{V}$ evaluates $\tilde{V}_x$ to check $\mathcal{P}$'s final claim about $\tilde{V}_d$. Checking a random linear combination would require $\mathcal{V}$ to evaluate $\tilde{V}_x$ at two points, whereas the reduce-from-two-points-to-one-point technique requires just one evaluation. Since evaluating $\tilde{V}_x$ is typically a bottleneck for the verifier [97, §3.3], eliminating the second evaluation is worthwhile even though it slightly increases the size of $\mathcal{P}$'s final message (and thus the proof size; see §4).

A detailed description of Giraffe is given in [97]. We give pseudocode for Gir^{++} in Appendix E. The efficiency and security of Gir^{++} are formalized in the following theorem, which can be proved via a standard analysis [51].

Theorem 1. *The interactive proof Gir^{++} satisfies the following properties when applied to a layered arithmetic circuit C of fan-in two, consisting of N identical sub-computations, each of depth d, with all layers of each sub-computation having width at most G. It has perfect completeness, and soundness error at most $((1 + 2\log G + 3\log N) \cdot d + \log G)/|\mathbb{F}|$. After a pre-processing phase taking time $O(dG)$, the verifier runs in time $O(|x| + |y| + d\log(NG))$, and the prover runs in time $O(|C| + d \cdot G \cdot \log G)$. If the sub-AC has a regular wiring pattern as defined in [36], then the pre-processing phase is unnecessary.*

4 Compiling Gir^{++} into a zero-knowledge proof

In this section, we explain a simple approach that compiles Gir^{++} (§3.2) into a (perfect) zero-knowledge argument of knowledge $\mathcal{Z}$ for AC satisfiability. In a nutshell, for a circuit satisfiability instance $(C(\cdot,\cdot), x, y)$, $\mathcal{Z}$ is obtained from Gir^{++} as follows. First, $\mathcal{P}$ commits to a w such that $C(x, w) = y$. Second, $\mathcal{P}$ and $\mathcal{V}$ use Gir^{++} on the circuit evaluation instance $C(x, w)$, except that $\mathcal{P}$ sends (hiding) *commitments* to its messages rather than sending the messages prescribed by Gir^{++}. Finally, using the homomorphism properties of the commitment scheme, $\mathcal{P}$ and $\mathcal{V}$ establish (in zero knowledge) that the values $\mathcal{P}$ committed to indeed satisfy all of checks that $\mathcal{V}$ performs in Gir^{++}. Because the result is a public-coin protocol, we can apply the Fiat-Shamir heuristic to make the protocol non-interactive (§7).

This section explains the basic approach; Sections 5 and 6 develop substantial efficiency improvements.

$\mathcal{Z}$ works as follows:

Step 0. $\mathcal{P}$ commits to the entries of $w \in \mathbb{F}^\ell$, and proves via the protocol of Section A that it knows how to open each commitment. Denote the commitments as $\mathsf{Com}(w_1), \ldots, \mathsf{Com}(w_\ell)$.

Step 1. $\mathcal{P}$ and $\mathcal{V}$ start running interactive proof Gir^{++}: $\mathcal{V}$ picks a random $(q', q_0) \in \mathbb{F}^{b_N} \times \mathbb{F}^{b_G}$ and computes $m_0 \leftarrow \tilde{V}_y(q', q_0)$, i.e., the multilinear extension of the outputs y evaluated at a random point (q', q_0). $\mathcal{V}$ sends (q', q_0) to $\mathcal{P}$. As per Step 1 of the basic IP Gir^{++}, $\mathcal{P}$ and $\mathcal{V}$ should apply the sum-check protocol to reduce the claim about $\tilde{V}_y(q', q_0)$ to a claim about $\tilde{V}_1(\cdot)$ at two points (r', r_0) and (r', r_1); however, $\mathcal{P}$ cannot directly use sum-check since it will not be zero knowledge.

Aside: Review of the sum-check protocol. Before explaining how to modify sum-check to ensure the entire protocol can be proven zero-knowledge, let us recall how this instance of sum-check proceeds. (We focus on the first invocation of the sum-check protocol in Gir^{++} only for notational convenience.)

In round one of the sum-check protocol, $\mathcal{P}$ sends a univariate polynomial $s_1(\cdot)$ of degree 3. $\mathcal{V}$ checks that $s_1(0) + s_1(1) = \tilde{V}_y(q', q_0)$, and then sends a random field element r_1 to $\mathcal{P}$. In general, in round j of the sum-check protocol, $\mathcal{P}$ sends a univariate polynomial s_j (which is degree 3 in the first b_N rounds and degree 2 in the remaining rounds). $\mathcal{V}$ checks that $s_j(0) + s_j(1) = s_{j-1}(r_{j-1})$, then sends a random field element r_j to $\mathcal{P}$. We write the vector of all r_j's chosen by $\mathcal{V}$ in the $j_{\text{last}} = b_N + 2b_G$ rounds of the sum-check protocol as $(r_1, \ldots, r_{j_{\text{last}}}) \in \mathbb{F}^{b_N+2b_G}$; let r' denote the first b_N entries of this vector, r_1 denote the next b_G entries, and r_2 denote the final b_G entries. In the final round, $\mathcal{P}$ sends $\mathcal{V}$ the claimed values v_0, v_1 of $\tilde{V}_1(r', r_1)$ and $\tilde{V}_1(r', r_2)$, and $\mathcal{V}$ checks that

$$\begin{aligned}
s_{j_{\text{last}}}(r_{j_{\text{last}}}) = \widetilde{\mathrm{eq}}(q', r') \cdot \big[&\tilde{\mathrm{add}}_i(q, r_0, r_1) \cdot (v_0 + v_1) + \\
&\tilde{\mathrm{mult}}_i(q, r_0, r_1) \cdot v_0 \cdot v_1\big]
\end{aligned}$$

Finally, $\mathcal{V}$ reduces $\mathcal{P}$'s claims about v_0, v_1 to an expression to which $\mathcal{V}$ and $\mathcal{P}$ inductively apply the sum-check (§3.2).

ZK variant of the sum-check protocol. The sum-check protocol just described is not zero knowledge: $\mathcal{P}$'s messages throughout the sum-check protocol, and in the "random linear combination" step, depend on the circuit's gate values as captured by the functions V_i for each layer i of C, and these gate values in turn depend on the witness w. To address this issue, we use a well-known "commit-and-prove" technique [39], thereby giving a zero-knowledge variant of the sum-check protocol.

In more detail, instead of sending the polynomial s_j to $\mathcal{V}$ "in the clear" at each round j of the sum-check protocol, $\mathcal{P}$ commits to the polynomial s_j and proves to $\mathcal{V}$ that it knows s_j. That is, for $s_j(t) = c_{3,j}t^3 + c_{2,j}t^2 + c_{1,j}t + c_{0,j}$, $\mathcal{P}$ sends commitments $\delta_{c_{3,j}} \leftarrow \mathsf{Com}(c_{3,j})$, $\delta_{c_{2,j}} \leftarrow \mathsf{Com}(c_{2,j})$, $\delta_{c_{1,j}} \leftarrow \mathsf{Com}(c_{1,j})$, and $\delta_{c_{0,j}} \leftarrow \mathsf{Com}(c_{0,j})$, then proves to $\mathcal{V}$ in zero knowledge that it knows how to open these commitments (e.g., with the protocol of Appx. A).

These commitments prevent the verifier from directly performing the required checks from Gir^{++}. Instead, $\mathcal{P}$ proves in zero knowledge that all of the verifier's checks "underneath the commitments" would pass. That is, in round 1 of the first invocation of the sum-check protocol in Gir^{++}, $\mathcal{V}$ needs to check that $s_1(0) + s_1(1) = \tilde{V}_y(q', q)$. $\mathcal{V}$ can compute a commitment to $s_1(0) + s_1(1) = c_{3,1} + c_{2,1} + c_{1,1} + 2c_{0,1}$ using the homomorphism property via $\delta_{c_{3,1}} \odot \delta_{c_{2,1}} \odot \delta_{c_{1,1}} \odot \delta_{c_{0,1}}^2$. $\mathcal{P}$ proves to $\mathcal{V}$ in zero knowledge that this is a commitment to $\tilde{V}_y(q', q)$ (e.g., with the protocol of Appx. A). Then $\mathcal{V}$ sends r_1 to $\mathcal{P}$, and computes a commitment U to $s_1(r_1)$ via the homomorphism, i.e., $U \leftarrow \delta_{c_{3,1}}^{r_1^3} \odot \delta_{c_{2,1}}^{r_1^2} \odot \delta_{c_{1,1}}^{r_1} \odot \delta_{c_{0,1}}$. In the next round, $\mathcal{V}$ computes a commitment $V = \mathsf{Com}(s_2(0) + s_2(1))$ from $\mathcal{P}$'s commitment to the coefficients of s_2, and $\mathcal{P}$ proves to $\mathcal{V}$ in zero knowledge that U and V are commitments to the same value (e.g., with the protocol of Appx. A).

In the final round j_{last} of the sum-check protocol, $\mathcal{V}$ computes a commitment W to $s_{j_{last}}(r_{j_{\text{last}}})$. $\mathcal{P}$ sends commitments X, Y, and Z to v_0, v_1, and $v_0 \cdot v_1$ respectively, and proves in zero knowledge (e.g., with the protocol of Appx. A.1) that the three committed values satisfy this product relation. $\mathcal{V}$ then computes a commitment Ω to

$$\tilde{\mathsf{eq}}(q', r') \cdot \left[\tilde{\mathsf{add}}_1(q, r_0, r_1)(v_0 + v_1) + \tilde{\mathsf{mult}}_1(q, r_0, r_1)(v_0 \cdot v_1)\right]$$

via

$$\Omega \leftarrow (X \odot Y)^{\tilde{\mathsf{eq}}(q', r') \cdot \tilde{\mathsf{add}}_1(r, r_1, r_2)} \odot Z^{\tilde{\mathsf{eq}}(q', r') \cdot \tilde{\mathsf{mult}}_1(r, r_1, r_2)}$$

$\mathcal{P}$ then proves in zero knowledge that W and Ω are commitments to the same value.

Finally, $\mathcal{V}$ computes a commitment to $\mu_0 v_0 + \mu_1 v_1$ via $X^{\mu_0} \odot Y^{\mu_1}$, which is what is needed for the next sum-check invocation (§3.2, "Random linear combination").

Final step. In the final step, $\mathcal{P}$ convinces $\mathcal{V}$ that the result of the final sum-check invocation is consistent with the computation's input and and witness. Specifically, recall (§3.2) that after the final invocation of the sum-check protocol in Gir^{++}, $\mathcal{P}$ uses the "reducing from two points to one point" technique, meaning that it sends a claim about $\tilde{V}_d$ restricted to a line; this is a univariate polynomial of degree at most $b_G = \log(G)$. $\mathcal{V}$'s task is to check that the claimed $\tilde{V}_d$ agrees with the input and witness. This is done in zero knowledge exactly as in each of the intermediate rounds of the sum-check protocol: $\mathcal{P}$ sends commitments to the coefficients of the univariate polynomial, proves knowledge of their openings, and then proves in zero knowledge that the verifier's check "underneath the commitments" would pass.

In more detail: $\mathcal{V}$ computes a commitment f to the evaluation of $\mathcal{P}$'s committed polynomial at the point $r = (r_0, \ldots, r_\ell) \in \mathbb{F}^{1+\ell}$. Since $\mathcal{P}$ claims this polynomial is is $\tilde{V}_d$ restricted to a line (and recalling that $\tilde{V}_d = \tilde{V}_x$), f must be equal to the multilinear extension of the computation's input and witness evaluated at the same point. Let $m = (x, w)$ denote the concatenation of the public input x and the witness w,[5] and assume for simplicity that they have the same length, say, $|x| = |w| = 2^\ell$. We interpret x, w, and m as functions (§3.2, "Multilinear extensions"). Because the MLE of m is unique (§3.2), it is easy to check that

$$\tilde{m}(r_0, r_1, \ldots, r_\ell) = (1 - r_0) \cdot \tilde{x}(r_1, \ldots, r_\ell) + r_0 \cdot \tilde{w}(r_1, \ldots, r_\ell)$$

Thus, $\mathcal{V}$'s commitment f must equal $\tilde{m}(r_0, \ldots, r_\ell)$. We now explain how the verifier can compute a second commitment to the same quantity using the commitments to w that $\mathcal{P}$ sent in Step 0 (above). We exploit the following standard expression for the multilinear extension of a function $w\colon \{0,1\}^\ell \to \mathbb{F}$:

$$\begin{aligned}\tilde{w}(r_1, \ldots, r_\ell) &= \sum_{b \in \{0,1\}^\ell} w(b) \cdot \prod_{k \in \{1,\ldots,\ell\}} \chi_{b_k}(r_k) \\ &= \sum_{b \in \{0,1\}^\ell} w(b) \cdot \chi_b \qquad (4)\end{aligned}$$

where $\chi_{b_k}(r_k) = r_k b_k + (1 - r_k)(1 - b_k)$, and $\chi_b = \prod_k \chi_{b_k}(r_k)$. For each $b \in \{0,1\}^\ell$, χ_b is the *Lagrange basis polynomial* for b.

$\mathcal{V}$ can compute, in the clear, $F' = (1 - r_0) \cdot \tilde{x}(r_1, \ldots, r_\ell)$. $\mathcal{V}$ can also compute a commitment F to $r_0 \cdot \tilde{w}(r_1, \ldots, r_\ell)$ from the commitment to each entry of w. Specifically, in linear time, $\mathcal{V}$ can evaluate each Lagrange basis polynomial χ_b at $(r_1, \ldots, r_\ell)$ [95], and then compute a commitment to $\tilde{w}(r_1, \ldots, r_\ell) = \sum_{b \in \{0,1\}^\ell} w(b) \cdot \chi_b(r_1, \ldots, r_\ell)$ as $\bigodot_{b \in \{0,1\}^\ell} \mathsf{Com}(w(b))^{\chi_b(r_1,\ldots,r_\ell)}$. A final exponentiation by r_0 gives F. With all of this in hand, $\mathcal{P}$ convinces $\mathcal{V}$ by proving in zero knowledge that f commits to the same value as $F \odot \mathsf{Com}(F'; 0)$ (e.g., with the protocol of Appx. A).

Theorem 2. *Let $C(\cdot, \cdot)$ be a layered arithmetic circuit of fan-in two, consisting of N identical sub-computations, each of depth d, with all layers of each sub-computation having width at most G. Assuming the existence of computationally binding, perfectly hiding homomorphic commitment schemes that support efficient zero-knowledge protocols for establishing knowledge of openings, equality of two commitments, and product relationships (Appx. A.1), there exists a PZK argument of knowledge for the NP relation "$\exists w$ such that $C(x, w) = y$". The protocol requires $d \log(G)$ rounds of communication, and has communication complexity $\Theta(|y| + (|w| + d \log G) \cdot \lambda)$, where λ is a security parameter. Given a w such that $C(x, w) = y$, the prover runs in time $\Theta(dNG + G \log G + (|w| + d \log G) \cdot \kappa)$ where κ is an upper bound on the time to compute a commitment, and run the zero-knowledge protocols for establishing knowledge of openings, equality of two commitments, and product relationships. The verifier runs in time $\Theta(|x| + |y| + (|w| + d \log(GN)) \cdot \kappa)$.*

The above follows from the more general Theorem (3.1) of [15].

[5] We regard the entire data-parallel computation (§3.2) as having one large input and one large witness; sub-ACs operate on disjoint "slices" of the large input and witness. We revisit this assumption in §6.2.

5 Reducing the cost of sum-checks

In the PZK argument from Section 4, the prover sends a separate commitment for every message element of Gir^{++} (§3.2), and then independently proves knowledge of how to open each commitment. This leads to long proofs and many expensive cryptographic operations for the verifier.

In this section, we explain how to reduce this communication and minimize the number of cryptographic operations for the verifier by exploiting *vector* or *multi*-commitment schemes, in which committing to a vector of elements at has essentially the same communication cost as committing to a single element. The celebrated Pedersen commitment (Appx. A), for example, can be made into a vector commitment scheme.

In such schemes, however, the additive homomorphism (§3.1) is over a vector rather than a scalar. Thus, given such a vector commitment to the coefficients a, b, d, e of the polynomial f, it is not possible for $\mathcal{V}$ to compute a commitment to $f(0) + f(1)$ and check equality as it did in the previous section. Instead, we show that using vector commitments, the Schwartz-Zippel lemma, and a dot-product proof protocol, $\mathcal{V}$ can verify *all* of $\mathcal{P}$'s claims from one sum-check invocation at once.

Dot-product proof protocol. Our starting point is an existing technique for designing efficient proof protocols for vector commitments. In this scenario, a prover knows the openings of commitments to a vector $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and a scalar $y \in \mathbb{F}$, and wants to prove in zero knowledge that $y = \langle \vec{a}, \vec{x} \rangle$ for a public $\vec{a} \in \mathbb{F}^n$. (This type of statement often appears in applications that use vector commitments.) The protocol and proof are given in Appendix A.2.

Lemma 3. *Let $\vec{x} = (x_1, \ldots, x_n)$ where $x_i \in \mathbb{F}$. Given a vector commitment $\mathsf{Com}(\vec{x})$, a scalar commitment $\mathsf{Com}(y)$, and a public vector $\vec{a} = (a_1, \ldots, a_n)$, the protocol of Appendix A.2 proves in zero knowledge that $y = \langle \vec{x}, \vec{a} \rangle$.*

Squashing $\mathcal{V}$'s checks. To exploit the dot-product proof protocol, we first recall from Section 4 that in each round j of each sum-check invocation in Gir^{++}, $\mathcal{P}$ sends commitments to $c_{3,j}$, $c_{2,j}$, $c_{1,j}$, and $c_{0,j}$, and proves to $\mathcal{V}$ that $c_{3,j} + c_{2,j} + c_{1,j} + 2c_{0,j} = s_{j-1}(r_{j-1})$ (i.e., that $s_j(0) + s_j(1) = s_{j-1}(r_{j-1})$). $\mathcal{V}$ then computes a commitment to $s_j(r_j) = c_{3,j} r_j^3 + c_{2,j} r_j^2 + c_{1,j} r_j + c_{0,j}$ for the next round.[6]

We observe that combining the above checks yields $c_{3,j+1} + c_{2,j+1} + c_{1,j+1} + 2c_{0,j+1} - (c_{3,j} r_j^3 + c_{2,j} r_j^2 + c_{1,j} r_j + c_{0,j}) = 0$. $\mathcal{V}$'s final check can also be expressed as a linear equation in terms of $v_0, v_1, c_{2,n}, c_{1,n}, c_{0,n}$, and wiring predicate evaluations (§3.2), $n = b_N + 2b_G$. We can therefore write $\mathcal{V}$'s checks during the rounds of the sum-check protocol as the matrix-vector product

$$\begin{bmatrix} M_1 \\ \vdots \\ M_{b_N+2b_G+1} \end{bmatrix} \cdot \vec{\pi} = \begin{bmatrix} s_0 \\ 0 \\ \vdots \end{bmatrix} \tag{5}$$

Each M_k is a row in $\mathbb{F}^{4b_N+6b_G+3}$ encoding one of $\mathcal{V}$'s checks and $\vec{\pi}$ is a column in $\mathbb{F}^{4b_N+6b_G+3}$ comprising $\mathcal{P}$'s messages. ($4b_N+6b_G+3$ accounts for b_N rounds with cubic s_j, $2b_G$ rounds with quadratic s_j, and the final values v_0, v_1, and $v_0 v_1$; §4.)

Now, by applying a standard Schwartz-Zippel technique, we can combine all of the linear equality checks encoded in Equation (5) into a single check, namely, by multiplying each row k by a random coefficient ρ_k and summing the rows.

Lemma 4. *For any $\vec{\pi} \in \mathbb{F}^\ell$, and any matrix $M \in \mathbb{F}^{n+1 \times \ell}$ with rows $M_1, \ldots, M_{n+1}$ for which Eq.* (5) *does not hold, then*

$$\Pr_{\rho}\left[\left\langle \left(\sum \rho_k \cdot M_k\right), \vec{\pi} \right\rangle = \rho_1 \cdot s_0\right] \leq 1/|\mathbb{F}|$$

Proof. Observe that $\left\langle (\sum \rho_k \cdot M_k), \vec{\pi} \right\rangle$ is a polynomial in $\rho_1, \ldots, \rho_{n+1}$ of total degree 1 (i.e., a linear function in $\rho_1, \ldots, \rho_{n+1}$). Call this linear polynomial ϕ. The coefficients of ϕ are the entries of $M \cdot \vec{\pi}$. Similarly, $\rho_1 \cdot s_0$ is a linear polynomial ψ in $\rho_1, \ldots, \rho_{n+1}$, whose coefficients are the entries of $[s_0, 0, \ldots, 0]$. Note that if Equation (5) does not hold, then ϕ and ψ are distinct polynomials, each of total degree 1. The lemma now follows from the Schwartz-Zippel lemma. □

Putting the pieces together. Lemma 4 implies that, once $\mathcal{P}$ has committed to $\vec{\pi}$, it can use Lemma 3 to convince $\mathcal{V}$ of the sum-check result in one shot. For soundness in Gir^{++}, however, $\mathcal{P}$ must commit to $c_{3,j}, c_{2,j}, c_{1,j}, c_{0,j}$ before the Verifier sends r_j. This means that $\mathcal{P}$ cannot send a single commitment to $\vec{\pi}$.

Instead, we observe that $\mathcal{P}$ can send the commitment to $\vec{\pi}$ *incrementally*, using one group element per round of the sum-check. That is, instead of committing to $\vec{\pi}$ all at once, $\mathcal{P}$ commits to one vector in each round of the sum-check encoding that round's polynomial coefficients. At the end of the sum-check, $\mathcal{V}$ checks $\mathcal{P}$'s messages using a modified version of Lemma 3, which we detail in Figure 1.

Lemma 5. *The protocol of Figure 1 establishes that its input values constitute an accepting sum-check relation. Specifically, on input a commitment C_0, commitments $\{\alpha_j\}$ to polynomials $\{s_j\}$ in a sum-check invocation, rows $\{M_k\}$ of the matrix of Equation* (5)*, and commitments $X = \mathsf{Com}(v_0)$, $Y = \mathsf{Com}(v_1)$, and Z, where $\{r_j\}$ are $\mathcal{V}$'s coins from the sum-check and $n{=}b_N + 2b_G$, the protocol establishes that $C_0{=}\mathsf{Com}(s_1(0){+}s_1(1))$; $s_j(0){+}s_j(1){=}s_{j-1}(r_{j-1})$, $j{\in}\{2, \ldots, n\}$; and $s_n(r_n){=}Q_{\ldots,i}$ evaluated with v_0, v_1 (per §3.2). This protocol is complete, honest-verifier perfect zero knowledge, and generalized special sound.*

Lemma 5 is proved in Appendix A.3. Relative to the PZK argument of Section 4, the protocol of Figure 1 reduces $\mathcal{P}$'s communication during a sum-check invocation by ≈3×. It also reduces $\mathcal{P}$'s and $\mathcal{V}$'s cryptographic costs by ≈4× and ≈5×, respectively (in part because it allows $\mathcal{P}$ and $\mathcal{V}$ to use multi-exponentiation [80]).

6 Reducing the cost of the witness

In the protocol of Section 4, $\mathcal{P}$'s initial message includes one commitment for each element $w_1, \ldots, w_n$ of the witness w (§4, "Step 0."). Then, at the end of the protocol, $\mathcal{P}$ convinces $\mathcal{V}$ that its final claim about $\tilde{V}_d$ is consistent with these initial commitments to w (§4, "Final step"). Unfortunately, this means that handling a circuit relation with $|w|$ witness elements requires a proof whose size is *at least* proportional to $|w|$.

[6] This description applies to the first b_N rounds, in which $\mathcal{P}$ sends a cubic polynomial. For rounds in which $\mathcal{P}$ sends a quadratic polynomial (i.e., the last $2b_G$ rounds), $c_{3,j} = 0$ and $\mathcal{P}$ does not send it.

EstablishSumCheckRelation($C_0, \{\alpha_j\}, \{M_k\}, X, Y, Z$)

Inputs: $C_0 = \mathsf{Com}(s_0; r_{C_0})$.

$\{\alpha_j\}$ are $\mathcal{P}$'s messages from the sum-check protocol: at each round j of the sum-check protocol, $\mathcal{P}$ has sent

$$\alpha_j \leftarrow \mathsf{Com}((c_{3,j}, c_{2,j}, c_{1,j}, c_{0,j}); r_{\alpha_j})$$

$\{M_k\}$ is defined as in Equation (5) and Lemma 4.

$X = \mathsf{Com}(v_0; r_X)$, $Y = \mathsf{Com}(v_1; r_Y)$, $Z = \mathsf{Com}(v_0 v_1; r_Z)$.

Definitions: $n = b_N + 2b_G$; $\vec{\pi}$ is defined as in Equation (5); $\{\rho_k\}$ are chosen by $\mathcal{V}$ (see below); $\vec{J} = \sum \rho_k \cdot \vec{M}_k$; (J_X, J_Y, J_Z) are the last 3 elements of $\vec{J}$; $\vec{\pi}^*$ and $\vec{J}^*$ are all but the last three elements of $\vec{\pi}$ and $\vec{J}$, respectively.

1. $\mathcal{P}$ proves in zero knowledge that the X, Y, and Z have the required relation (e.g., with the protocol of Appx. A.1).
2. $\mathcal{P}$ picks $r_{\delta_1}, \ldots, r_{\delta_n} \in_R \mathbb{F}$ and $\vec{d} \in_R \mathbb{F}^{4b_N + 6b_G}$ where $\vec{d} = (d_{c_{3,1}}, d_{c_{2,1}}, d_{c_{1,1}}, d_{c_{0,1}}, \ldots, d_{c_{0,n-1}}, d_{c_{2,n}}, d_{c_{1,n}}, d_{c_{0,n}})$. $\mathcal{P}$ computes and sends
$$\delta_j \leftarrow \mathsf{Com}((d_{c_{3,j}}, d_{c_{2,j}}, d_{c_{1,j}}, d_{c_{0,j}}); r_{\delta_j}), \quad j \in \{1, \ldots, n\}$$
3. $\mathcal{V}$ chooses and sends $\rho_1, \ldots, \rho_{n+1} \in_R \mathbb{F}$.
4. $\mathcal{P}$ picks $r_C \in_R \mathbb{F}$, then computes and sends
$$C \leftarrow \mathsf{Com}(\langle \vec{J}^*, \vec{d} \rangle; r_C)$$
5. $\mathcal{V}$ chooses and sends challenge $c \in_R \mathbb{F}$.
6. $\mathcal{P}$ computes and sends
$$\begin{aligned} \vec{z} &\leftarrow c \cdot \vec{\pi}^* + \vec{d} \\ z_{\delta_j} &\leftarrow c \cdot r_{\alpha_j} + r_{\delta_j}, \quad j \in \{1, \ldots, n\} \\ z_C &\leftarrow c \cdot \left(\rho_1 r_{C_0} - J_X r_X - J_Y r_Y - J_Z r_Z\right) + r_C \end{aligned}$$
7. $\mathcal{V}$ rejects unless the following holds, where we denote $\vec{z} = (z_{c_{3,1}}, z_{c_{2,1}}, z_{c_{1,1}}, z_{c_{0,1}}, \ldots, z_{c_{0,n-1}}, z_{c_{2,n}}, z_{c_{1,n}}, z_{c_{0,n}})$:
$$\mathsf{Com}((z_{c_{3,j}}, z_{c_{2,j}}, z_{c_{1,j}}, z_{c_{0,j}}); z_{\delta_j}) \stackrel{?}{=} \alpha_j^c \odot \delta_j \quad j \in \{1, \ldots, n\}$$
$$(C_0^{\rho_1} \odot X^{-J_X} \odot Y^{-J_Y} \odot Z^{-J_Z})^c \odot C \stackrel{?}{=} \mathsf{Com}(\langle \vec{J}^*, \vec{z} \rangle; z_C)$$

FIGURE 1—A ZK proof that captures $\mathcal{V}$'s checks in one sum-check invocation. $\mathcal{P}$ uses a modified version of Lemma 3 to convince $\mathcal{V}$ of the statement derived by applying Lemma 4. Values corresponding to $c_{3,j}$ are elided for all sum-check rounds j having quadratic s_j.

In this section, we describe a protocol that leverages vector commitments to reduce witness commitment size (and thus proof size) to sub-linear in $|w|$. This protocol uses a "matrix commitment" scheme due to Groth [54] that also reduces $\mathcal{V}$'s *computation cost* to sub-linear in $|w|$. We first describe this protocol restricted to purely data-parallel computations, meaning that each sub-AC has its own inputs and witness elements (§6.1; we assumed the same in §4, Fn. 5). We then remove this restriction in an efficient way, introducing a *redistribution layer* that allows input and witness sharing among sub-ACs (§6.2).

6.1 Details of improved witness handling

Recall (§4) that $m = (x, w)$ is the concatenation of the public input x and the witness w. As in Section 4, we assume that each sub-AC of the data-parallel computation operates on a disjoint "slice" of x and w, and we assume WLOG (purely for notational convenience) that $2^\ell = |x| = |w|$.

$\mathcal{V}$'s last step (§4, "Final step") is to check that

$$\begin{aligned} &\mathsf{Com}(\tilde{V}_d(r_0, \ldots, r_\ell)) \stackrel{?}{=} \\ &\qquad \mathsf{Com}(\tilde{V}_w(r_1, \ldots, r_\ell))^{r_0} \odot (1 - r_0)\tilde{V}_x(r_1, \ldots, r_\ell) \end{aligned} \quad (6)$$

To do this, $\mathcal{V}$ must compute a commitment to $\tilde{w}(r_1, \ldots, r_\ell)$, i.e., a commitment to the dot product of $(w_1, \ldots, w_{2^\ell})$ with $(\chi_1, \ldots, \chi_{2^\ell})$ by Equation (4). Naively, this seems to indicate that $\mathcal{P}$ should send one vector commitment to w in step 0, then prove the dot product in zero knowledge (e.g., with the protocol of Appx. A.2) in the final step. Unfortunately, this would not reduce $\mathcal{P}$'s communication: the dot product protocol has cost proportional to the size of the committed vector (because of $\vec{z}$), so proof size would still be proportional to $|w|$.

However, we observe that the coefficients χ_b in the dot product of Equation (4) have a special form that lets us adapt a protocol due to Groth [54] for the witness commitment.[7] Specifically, the vector $(\chi_1, \ldots, \chi_{2^\ell})$ can be computed by taking the tensor product of two smaller vectors.[8] Defining $\check{\chi}_b = \prod_{k=0}^{\ell/2} \chi_{b_k}(r_k)$ (which computes over the "lower bits" of b) and $\hat{\chi}_b = \prod_{k=\ell/2+1}^{\ell} \chi_{b_k}(r_k)$ (which computes over the "upper bits" of b), we have that

$$\chi_b = \prod_{k=0}^{\ell} \chi_{b_k}(r_k) = \left(\prod_{k=0}^{\ell/2} \chi_{b_k}(r_k)\right)\left(\prod_{k=\ell/2+1}^{\ell} \chi_{b_k}(r_k)\right) = \check{\chi}_b \cdot \hat{\chi}_b$$

Letting $\ell_2 = 2^{\ell/2} = \sqrt{|w|}$, we can rewrite Equation (4) as

$$\begin{aligned} &\tilde{V}_w(r_1, \ldots, r_\ell) = \\ &\begin{bmatrix} \check{\chi}_0 & \cdots & \check{\chi}_{\ell_2 - 1} \end{bmatrix} \begin{bmatrix} w_0 & w_{\ell_2} & \cdots & w_{|w|-\ell_2} \\ w_1 & w_{\ell_2+1} & \cdots & w_{|w|-\ell_2+1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\ell_2-1} & w_{2\ell_2-1} & \cdots & w_{|w|-1} \end{bmatrix} \begin{bmatrix} \hat{\chi}_0 \\ \hat{\chi}_{\ell_2} \\ \vdots \\ \hat{\chi}_{\ell_2(\ell_2-1)} \end{bmatrix} \\ &= L \cdot T \cdot R \end{aligned}$$

To see why this is true, notice that element $(i + 1, j + 1)$ of the matrix T is $w_{i+\ell_2 \cdot j}$, and further that this element is multiplied by $\check{\chi}_i \cdot \hat{\chi}_{\ell_2 \cdot j} = \chi_{i+\ell_2 \cdot j}$ by the definitions of $\check{\chi}$ and $\hat{\chi}$.

This suggests the following protocol: $\mathcal{P}$ first commits to each *row* of T using vector commitments. After all sum-check invocations in Gir^{++}, $\mathcal{V}$ checks $\mathcal{P}$'s claim about $\tilde{V}_d(r_0, \ldots, r_\ell)$ by computing the product of vector L (which the verifier can compute itself using $O(\sqrt{|w|})$ operations) with the commitments $T_1, \ldots, T_{\ell_2}$. The resulting commitment, $T' = \bigodot_{k=0}^{\ell_2} T_k^{\check{\chi}_k}$, is a vector commitment, and $\mathcal{P}$ proves in zero knowledge that the

[7] See [27] for a related application of this idea in the context of univariate rather than multivariate polynomials.

[8] This was also observed by Wahby et al. [97, §3.3], who used this fact to design a fast algorithm for evaluating the Lagrange basis polynomials, $\{\chi_b\}$.

dot product of the vector committed in T' with the vector R is $\tilde{V}_d(r_0, \ldots, r_\ell) - (1 - r_0)\tilde{V}_x(r_1, \ldots, r_\ell)$ (e.g., with the protocol of Appx. A.2). Since the size of the vector in T' contains $\sqrt{|w|}$ elements, the size of this proof and $\mathcal{V}$'s computational costs are both $\sqrt{|w|}$.

Lemma 6. *Without loss of generality, assume $|w| = 2^{2\ell'}$. Suppose that $\mathcal{P}$ commits to $w \in \mathbb{F}^{2^{2\ell'}}$ as described above using $2^{\ell'} = \sqrt{|w|}$ vector commitments. Later, for any $r = (r_1, \ldots, r_{2\ell'})$, $\mathcal{P}$ can send a commitment α and prove in zero knowledge that α commits to $\tilde{V}_w(r_1, \ldots, r_{2\ell'})$ using the protocol of Appendix A.2 in communication $\Theta(\sqrt{|w|})$, where $\mathcal{V}$ runs in $\Theta(\sqrt{|w|})$ steps. This protocol is complete, honest-verifier perfect zero knowledge, and generalized special sound.*

The completeness and zero knowledge of this protocol follow from the analysis of the protocol in Appendix A.2. We provide an analysis for generalized special soundness in Appendix A.4.

Further reducing witness cost. The witness commitment cost can be further reduced, at the cost of additional computation for $\mathcal{P}$ and $\mathcal{V}$. Briefly, $\mathcal{P}$ sends one vector commitment to w at the start of the protocol. After all sum-check invocations in Gir^{++}, $\mathcal{P}$ and $\mathcal{V}$ can compute $\mathcal{V}$'s final check (Eq. (6)) by adapting the recursive reduction due to Bootle et al. [27, §4]. (Their technique computes the public value of the inner product of two committed vectors, while we require a statement like Lemma 3, where one vector and the purported output are committed and the other vector is public; the adaptation is straightforward.) This has communication cost logarithmic in $|w|$, but requires $\mathcal{V}$ and $\mathcal{P}$ to do cryptographic operations linear in $|w|$. Exploring this tradeoff is future work.

6.2 Sharing witness elements in the data-parallel setting

We have thus far regarded the computation as having one large input and one large witness. When evaluating a data-parallel computation, this means that the sub-ACs' inputs must be disjoint slices of the full input (and similarly for the witness). Unfortunately, this is not sufficient in many cases of interest.

Consider the example of a Merkle tree: $\mathcal{P}$ wants to convince $\mathcal{V}$ that it knows the leaves of a Merkle tree corresponding to a supplied root. Verifying a witness with M leaves requires $2M-1$ invocations of a hash function, which naturally corresponds to a computation with $2M-1$ sub-ACs laid side-by-side, each encoding a hash function invocation.[9] But this means that sub-AC outputs must feed other sub-AC inputs, which requires sub-ACs to *share witness elements*. (We note that simply duplicating values in the witness is not a solution because a cheating $\mathcal{P}$ can equivocate, giving different values to different sub-ACs.)

One solution is a hybrid vector-scalar scheme: $\mathcal{P}$ supplies individual commitments for each shared witness element, plus one set of vector commitments (as in the prior section) for the non-shared elements. This works because it allows $\mathcal{V}$ to use scalar commitments for arbitrary inputs: given a commitment δ, $\mathcal{V}$ can "inject" the committed value into index b by multiplying the right-hand side of Equation (14) (Appx. A.2) by δ^{χ_b}.

[9]The sub-ACs could be arranged sequentially rather than side-by-side, which would avoid the issues described in this subsection. But this would drastically increase circuit depth, and hence the proof length and associated costs when applying our argument to the resulting circuit.

The above approach works when the number of shared witness elements is small, but it is inefficient when there are many shared elements, because each shared element requires a separate commitment and proof of knowledge. For such cases, we enable sharing of witness elements using a special modification to the arithmetic circuit encoding the NP relation. Specifically, after constructing a data-parallel AC corresponding to the computation, we add one non-data-parallel *redistribution layer* (RDL) whose inputs are the full input and witness, and whose outputs feed the input layers of each sub-AC. Since this layer is not data parallel, there are no restrictions on which inputs it can copy to which outputs, so $\mathcal{V}$ can use the RDL to copy the same witness element to multiple sub-ACs. And because the protocol forces $\mathcal{P}$ to respect the wiring pattern of the RDL, $\mathcal{P}$ cannot equivocate about the witness values.

Moreover, since this layer only "re-wires" the inputs, the corresponding sum-check invocation can be optimized to require fewer rounds and a simplified final check. Observe that the redistribution layer only requires one-input "pass" gates that copy their input to their output. Thus, following a simplification of the CMT protocol [36, 91], we have that

$$\tilde{V}_{d-1}(q', q) = \sum_{h \in \{0,1\}^{\log(|m|)}} \tilde{\text{pass}}((q', q), h) \cdot \tilde{V}_d(h)$$

where $\tilde{\text{pass}}((q', q), r)$ is the MLE of a wiring predicate (§3.2) that evaluates to 1 when the RDL connects from the AC input with label h to input q in sub-AC q' (and 0 otherwise). A sum-check over $\text{RDL}_{(q',q)}(h) = \tilde{\text{pass}}((q', q), h) \cdot \tilde{V}_d(h)$ requires $\log(|m|) = \log(|x| + |w|)$ rounds and ends with a claim about a *single point* in $\tilde{V}_d$—so $\mathcal{V}$ can check this claim (using the protocol of §6.1) *without* using either "reducing from two points to one point" or the "random linear combination" technique.

$\mathcal{V}$'s primary cost related to the RDL is evaluating $\tilde{\text{pass}}((q', q), r_0)$, where $r_0 \in \mathbb{F}^{\log(|m|)}$ is a vector of $\mathcal{V}$'s random coins from the sum-check. This costs $O(|m| + NG)$ via known techniques [97, §3.3].

7 Hyrax: a zkSNARK based on Gir^{++}

We refer to the PZK argument obtained by combining the protocol of Section 5 with the improved witness-commitment protocol of Section 6 as Hyrax-Interactive; full pseudocode is given in Appendix D. Since Hyrax-Interactive is a public-coin protocol, we apply the Fiat-Shamir heuristic [42] to produce a zkSNARK that we call Hyrax.

Implementation. We have built a proof-of-concept implementation of Hyrax based on Giraffe [1, 97]. Our implementation uses Pedersen commitments in an elliptic curve group (Appx. A) of order $q_{\mathcal{G}}$ and works with ACs over $\mathbb{F}_{q_{\mathcal{G}}}$. We instantiate the random oracle with SHA-256.

The prover takes as input a high-level description of a computation (in the same format produced by Giraffe's C compiler), the public inputs, and an auxiliary executable that generates the witness from the public inputs; the prover's output is a proof. The verifier takes as input the same computation description and public inputs plus the proof, and outputs "accept" or "reject."

We added about 2000 lines of Python to Giraffe to implement the "random linear combination" technique (§3.2),

commitments, and non-interactive proof generation, serialization, and deserialization. We also added support for elliptic curve primitives to Python via the Python API [6]; this comprises 1500 lines of C providing an interface to the MIRACL Crypto SDK [4]. MIRACL supports many curves; we have tested Curve25519 [24], M221, M191, and M159 [11]. We produce random generators for commitments by hashing, which we built in 200 lines of Sage [7], adapting a script by Aranha et al. [11].

We emphasize that our implementation is a proof of concept, and not optimized for speed. For example, it uses Python's native multiprecision integer type for field elements, meaning that essentially all field operations involve memory allocation (which is slow). Furthermore, our elliptic curve primitives' flexibility comes at a price: in our informal testing, Curve25519 is more than 10× slower for us than for the fastest implementations [3]. Careful cost modeling (§8.2) suggests that an optimized implementation will be faster by an order of magnitude or more.

8 Evaluation

In this section we ask:

- Is Hyrax concretely efficient for realistic computations?
- How does Hyrax compare to several baseline systems, considering proof size and $\mathcal{V}$ and $\mathcal{P}$ execution time?
- How do Hyrax's refinements (§5–6) affect its costs?

We find that our implementation can handle up to 27× larger computations than the highly-optimized libsnark [9] with comparable resources [98]. We also find that Hyrax's refinements result in significant savings, and that Hyrax produces proofs comparable to or smaller than all but one of the baseline protocols, for problem sizes that our implementation can handle. Finally, because our implementation is in Python, it is slow; but by carefully accounting costs in terms of basic operations, we find that Hyrax has the lowest cost for $\mathcal{V}$ and the second-lowest for $\mathcal{P}$ among the baselines. Future work is making these advantages concrete with a fast implementation.

Benchmarks. We evaluate using three benchmarks:

• *Matrix multiplication* proves to $\mathcal{V}$ that $\mathcal{P}$ knows two matrices whose product equals the public input. We evaluate on 16×16, 32×32, 64×64, and 128×128 matrices, and for each we vary N, the number of parallel executions.[10]

• *Image scaling* establishes that $\mathcal{V}$'s input, a low-resolution image, is a scaled version of a high-resolution image that $\mathcal{P}$ knows. For scaling, we use Lanczos resampling [93], which is a standard image transformation in which each output pixel is the result of convolving a two-dimensional windowed sinc function [76] with the input image.

We evaluate on 4×, 16×, 64×, and 256× scaling, varying the number of pixels. This is a data-parallel computation where each sub-computation evaluates one pixel of the low-resolution image, but pixels of the high-resolution image must be shared between sub-computations corresponding to neighboring pixels in the low-resolution image. To accommodate this, we use a redistribution layer (RDL; §6.2).

• *Merkle tree* proves to $\mathcal{V}$ that $\mathcal{P}$ knows an assignment to the leaves of a Merkle tree [72] corresponding to a Merkle root, which $\mathcal{V}$ provides. We use SHA-256 for the hash, varying the number of leaves in the tree; we implement a data-parallel computation in which each sub-computation is one invocation of SHA-256; and we connect outputs at one level of the tree to inputs at the next level using a redistribution layer. For M leaves, the benchmark comprises $2M-1$ sub-computations.

To implement SHA-256 efficiently in an arithmetic circuit, we use an approach from prior work [19] for efficient addition modulo 2^{32}. We describe the approach, and an optimization that may be of independent interest, in Appendix C.

8.1 Scaling and concrete efficiency

Setup and method. We run our experiments on Amazon Elastic Compute Cloud (EC2) [2] c3-4xlarge instances, which have 30 GiB of RAM and 16 Intel Xeon E5-2680v2 vCPUs (8 cores, 2 hyperthreads per core) running at 2.8 GHz. The exception is the largest Lanczos experiment (Figs. 2c and 2d), for which we use a c3-8xlarge instance with 60 GiB of RAM and 32 vCPUs. (Only RAM is relevant to our experiments because our implementation is single threaded.)

For each benchmark, we construct a set of arithmetic circuits (and, for image scaling and Merkle trees, RDLs) for the parameters described above. $\mathcal{G}$ is M191 [11], an elliptic curve over a base field modulo $2^{191}-19$ with a subgroup of order $q_{\mathcal{G}} = 2^{188}+785475793603776042559315454 1$, for ≈90-bit security. ACs are over $\mathbb{F}_{q_{\mathcal{G}}}$, and group elements and scalars are 24 bytes. We prove and verify for each AC, measuring time using the high-resolution system clock and recording proof size.

Results. Figure 2 shows proof size and execution time for each benchmark. (We show 64×64 matrices and 16× image scaling; others are similar.) Proof size tracks the square root of computation size: witness size $|w|$ tracks computation size, witness commitment cost tracks $\sqrt{|w|}$ (§6), and this cost dominates.

$\mathcal{P}$ times are roughly linear with computation size. For matrix multiplication, the sum-check verifications dominate $\mathcal{V}$'s time for small N; for large N, $\mathcal{V}$ time tracks N because it is dominated by evaluating $\tilde{V}_y$ (§3.2). In the other two benchmarks, the RDL (§6.2) dominates $\mathcal{V}$'s costs at large sizes.

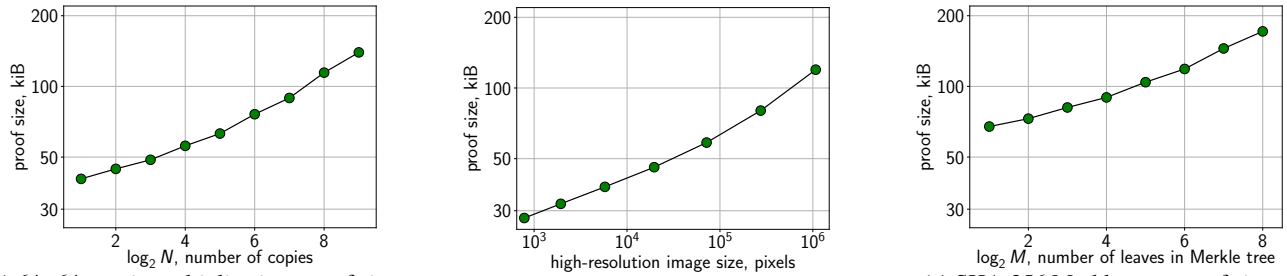
Our implementation is slow because we use Python, but Hyrax nevertheless scales to large computations because Gir^{++} is highly efficient. Notably, the largest matrix multiplication and Merkle tree computations comprise ≈270 million and ≈60 million gates, respectively 27× and 6× larger ACs than libsnark [9] handles on similar hardware [98].

8.2 Comparison with prior work

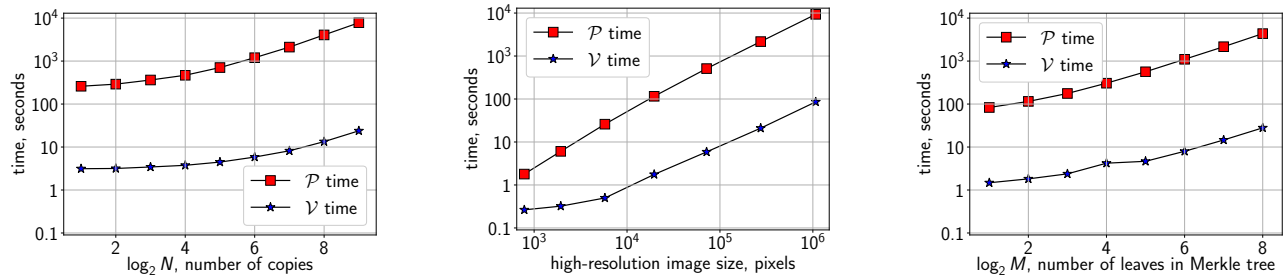
We now compare Hyrax's proof size and $\mathcal{V}$ and $\mathcal{P}$ cost with several baseline systems. Because we do not have comparable implementations of these systems, we use cost modeling and reported results. Section 8.3 discusses limitations of this analysis.

Baselines. We consider four state-of-the-art zero-knowledge argument systems with similar properties to Hyrax's: two argument systems due to BCCGP [27], which we call BCCGP-sqrt

[10] Matrix multiplication is a popular benchmark, but it is a poor application for general-purpose proof systems because it has fast, special-purpose protocols [46, 90]. We nevertheless include it, both for comparison to prior work [27, 78, 98] and as a generic example of arithmetic computation.

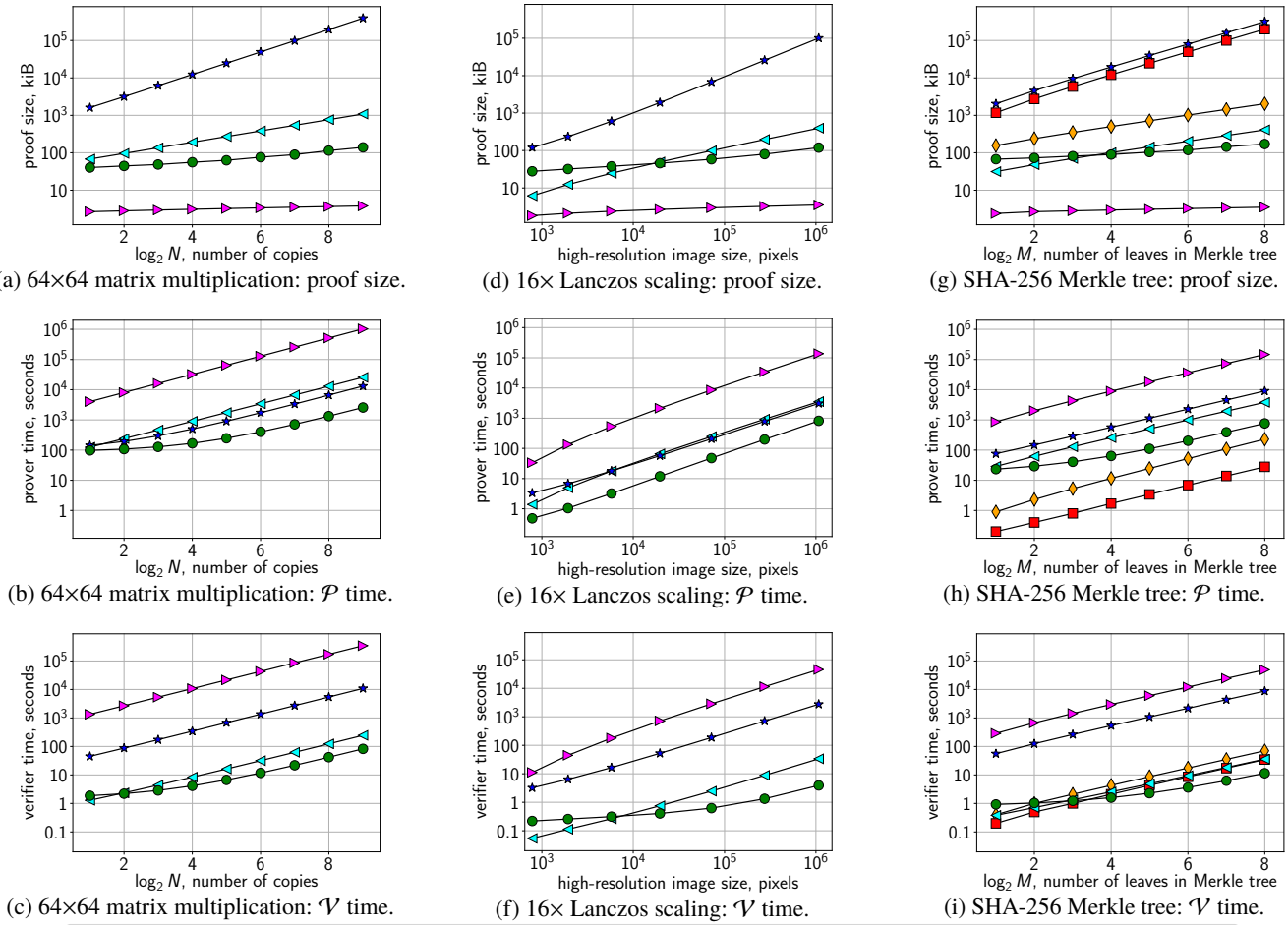


(a) 64×64 matrix multiplication: proof size. (c) 16× Lanczos scaling: proof size. (e) SHA-256 Merkle tree: proof size.



(b) 64×64 matrix multiplication: $\mathcal{P}$ and $\mathcal{V}$ time. (d) 16× Lanczos scaling: $\mathcal{P}$ and $\mathcal{V}$ time. (f) SHA-256 Merkle tree: $\mathcal{P}$ and $\mathcal{V}$ time.

FIGURE 2—Hyrax scaling (§8.1). We evaluate on matrix multiplication, Lanczos image scaling, and Merkle trees. In all three applications, Hyrax's proofs scale with the square root of the witness size. For matrix multiplication, Hyrax scales to ≈270 million gate ACs using less than 30 GiB of RAM. For image scaling and Merkle trees, $\mathcal{V}$ computation costs are dominated by the RDL (§6.2) at large computation sizes.



(a) 64×64 matrix multiplication: proof size. (d) 16× Lanczos scaling: proof size. (g) SHA-256 Merkle tree: proof size.

(b) 64×64 matrix multiplication: $\mathcal{P}$ time. (e) 16× Lanczos scaling: $\mathcal{P}$ time. (h) SHA-256 Merkle tree: $\mathcal{P}$ time.

(c) 64×64 matrix multiplication: $\mathcal{V}$ time. (f) 16× Lanczos scaling: $\mathcal{V}$ time. (i) SHA-256 Merkle tree: $\mathcal{V}$ time.

FIGURE 3—Comparison of concrete costs between the baseline systems and Hyrax (§8.2). BCCGP-sqrt, BCCGP-log [27], ZKB++ [32], and Ligero [10] are prior work; Hyrax-unopt is Hyrax without the optimizations of Sections 5–6.

and BCCGP-log; ZKB++ [32]; and Ligero [10]. We also consider a "naive" version of Hyrax, Hyrax-unopt, that uses the protocol of Section 4 without the refinements of Sections 5–6. We do not compare to systems that require trusted setup (e.g., libsnark [23]), but we discuss such systems briefly in Section 8.3.

Method. We compare Hyrax and the baseline systems on proof size and $\mathcal{P}$ and $\mathcal{V}$ time for the benchmark computations, except that for ZKB++ and Ligero, we only compare SHA-256 (§8.3).

We estimate costs for each system as follows:

- *Hyrax and Hyrax-unopt*: we adapt the Giraffe cost model [97, Appx. C] and carefully analyze the cost of compiling to a zkSNARK. We check the computational cost models by instrumenting our implementation to count basic operations; in other words, the model estimates the performance of an optimized implementation of Hyrax. For proof sizes in Hyrax, we use the values experimentally determined in Section 8.1; for Hyrax-unopt, the model reports proof size.
- *BCCGP-sqrt and -log*: we use the reported cost models [27].
- *ZKB++ and Ligero*: we estimate costs by extrapolating from reported performance on SHA-256 [10, 32, 94]. For ZKB++, we assume that execution times are the same as in ZKBoo [48]; this is optimistic for $\mathcal{V}$ because ZKB++ requires more work.

We use the arithmetic circuits from Section 8.1 as inputs to the models for Hyrax, Hyrax-unopt, BCCGP-sqrt, and BCCGP-log, using M191 [11] for $\mathcal{G}$ as in Section 8.1. For ZKB++ and Ligero, we estimate costs assuming 2^{-80} soundness error.[11]

To establish the time required for basic operations in Hyrax, Hyrax-unopt, BCCGP-sqrt, and BCCGP-log, we run each operation 2^{14} times, measuring average time using the high-resolution system clock on the same hardware as in Section 8.1. We use MIRACL [4] for elliptic curve operations, NTL [88] for field operations, and OpenSSL [5] for random number generation.

Results. Figure 3 compares costs for the 64×64 matrix multiplication, 16× image scaling, and Merkle tree benchmarks (other problem sizes are similar). For proof size, (Figs. 3a, 3d, 3g), Hyrax is worse both asymptotically and concretely than BCCGP-log. For large enough computations Hyrax's proofs are smaller than BCCGP-sqrt's, because Hyrax's cost tracks $\sqrt{w}$ asymptotically, while BCCGP-sqrt's tracks $\sqrt{\mathcal{M}}$ for an AC with $\mathcal{M}$ multiplication gates; on matrix multiplication, where $|w| \ll |\mathcal{M}|$, Hyrax has much smaller proofs. Hyrax's proofs are concretely smaller than Ligero's because the latter's cost tracks $\sqrt{|C|}$ for AC C. ZKB++'s cost is linear in the number of AND gates and Hyrax-unopt's cost tracks $|w|$; both are large.

Hyrax has lower $\mathcal{P}$ cost (Figs. 3b, 3e, 3h) than BCCGP-sqrt and BCCGP-log because it does cryptographic operations only for $\mathcal{P}$'s messages in Gir^{++} and for w (§4–§6). BCCGP-sqrt and BCCGP-log require cryptographic operations quasi-linear and linear in $\mathcal{M}$, respectively (BCCGP-sqrt's cost is worse asymptotically but better concretely at these problem sizes). Hyrax's $\mathcal{P}$ is concretely more expensive than ZKB++ or Ligero because those systems do not use expensive public-key cryptography. Ligero is asymptotically more costly than Hyrax, but this is not apparent at the problem sizes we consider. Hyrax's refinements compared to Hyrax-unopt (§5–6) result in much lower cost.

Hyrax has lower $\mathcal{V}$ cost (Figs. 3c, 3f, 3i) than BCCGP-sqrt for large enough computations, because its asymptotically dominant cost is non-cryptographic operations (§8.1). BCCGP-log has particularly high cost: its verifier does several cryptographic operations for each multiplication gate in the AC. ZKB++ and Ligero have lightweight verifiers, but their cost is linear in the computation's size. In contrast, Hyrax's $\mathcal{V}$ can be sublinear in some cases (§3.2), but this is not the case for the Merkle tree because of the cost of the RDL (§6.2). Hyrax-unopt requires cryptographic operations proportional to $|w|$; Hyrax's refinements reduce this to $\left|\sqrt{w}\right|$ for up to $10^3\times$ savings.

8.3 Discussion

Our results suggest that Hyrax is both concretely efficient and competitive with the baseline systems. First, Hyrax scales to computations 6–27× larger than the highly-optimized libsnark [9] for similar hardware [98], because the underlying proof system is highly efficient. Second, Hyrax's proofs are generally smaller than all but BCCGP-log, and the latter pays for its smaller proofs with high computational costs. Finally, while our current implementation is slow, our modeling suggests that an optimized implementation will be much faster—especially because prior work suggests that Gir^{++} is highly parallelizable [92, 96, 97].

On the other hand, there are several limitations to this analysis. First, Hyrax is competitive with prior work primarily when computations have sufficient data parallelism or are amenable to batching. We have demonstrated that an RDL (§6.2) lets Hyrax take advantage of parallelism within one computation (e.g., the Merkle tree). Even so, not all applications fit these paradigms.

Second, our comparison relies on models and microbenchmarks. We note that our $\mathcal{P}$ and $\mathcal{V}$ cost estimates for Hyrax and the BCCGP systems account only for basic operations (and not, say, memory allocation), so these are lower bounds; in contrast, ZKB++ and Ligero costs come from measurements on built systems, and are thus upper bounds. This means that ZKB++ and Ligero may perform better relative to other systems than predicted. Ongoing work is an optimized implementation in C++, which will make these comparisons more concrete.

Third, we have only compared against ZKB++ and Ligero on SHA-256. This makes sense for ZKB++ because it is best suited to Boolean circuits, and SHA-256 is a natural benchmark. Ligero's primary evaluation is also on SHA-256 [10, §6], but future work is evaluating its performance on ACs over large fields like the matrix multiplication and image scaling benchmarks. We note that, since Hyrax's proof size is primarily due to witness size $|w|$ rather than computation size $|C|$, it will outperform Ligero on applications like matrix multiplication where $|w| \ll |C|$.

Finally, our comparison does not consider argument systems like libsnark [9, 23] that require trusted setup and non-standard, non-falsifiable assumptions (§2), because Hyrax's goal is to avoid these requirements. Ignoring this goal, Hyrax's proof size is worse: libsnark gives constant-size proofs. Hyrax's $\mathcal{P}$ cost is concretely and asymptotically better: libsnark has a logarithmic overhead in $|C|$, and it requires cryptographic operations for each

[11] ZKB++ and Ligero give statistical security while the other systems make computational assumptions; this makes apples-to-apples comparison difficult. We have chosen parameters to give all systems roughly equivalent cost to prove a false statement assuming the best-known attacks against ECDH.

AC gate, while Hyrax's $\mathcal{P}$ is essentially linear in computation size and requires cryptographic operations only for $\mathcal{P}$'s messages in Gir^{++} and for w (§4–§6). For $\mathcal{V}$'s costs, libsnark's offline setup is very expensive (and it must be performed by $\mathcal{V}$ or someone $\mathcal{V}$ trusts), but its online costs are essentially always cheaper than Hyrax's. A careful comparison is future work.

9 Conclusion

We have described a succinct zero-knowledge argument for NP with no trusted setup and low concrete cost for both the prover and the verifier, based on standard cryptographic assumptions. This scheme is practical because it tightly integrates three components: a state-of-the-art interactive proof (IP), which we tweak to reduce communication complexity; a highly optimized transformation from IPs to zero-knowledge arguments following the approach of Ben-Or et al. [15] and Cramer and Damgård [39]; and a new cryptographic commitment scheme for multilinear polynomials that we tailor to the structure of the IP.

Our multilinear polynomial commitment scheme adapts prior work [54] and is based on standard cryptographic assumptions. The scheme allows a sender to commit to a log G-variate multilinear polynomial and later to open it at one point, with $\sqrt{G}$ total communication and receiver computation. A key direction for future work is further reducing communication cost without significantly increasing runtime.

More broadly, our work suggests that the GKR interactive proof [51] may be underappreciated as a building block for (zero-knowledge) arguments in which the verifier runs in time sublinear in the size of a circuit verifying the NP relation. In particular, the GKR protocol only saves work for the verifier for low-depth circuits, which might appear to limit its usefulness for general-purpose computations. But combining the GKR protocol with a *succinct* polynomial commitment scheme sidesteps this limitation, because it allows one to apply known general-purpose transformations from programs running in time T to instances of arithmetic circuit satisfiability in which the circuit has size $\tilde{O}(T)$ and depth $O(\log T)$—and for such arithmetic circuits, the GKR verifier *can* save work.

Acknowledgments

This work was funded by DARPA grant HR0011-15-2-0047 and NSF grants CNS-1423249, TWC-1646671, and TWC-1664445; and by Nest Labs and a Google Research Fellowship. Justin Thaler was supported by a Research Seed Grant from Georgetown University's Massive Data Institute.

References

[1] https://github.com/pepper-project.
[2] AWS EC2. https://aws.amazon.com/ec2/instance-types/.
[3] eBATS. http://bench.cr.yp.to/ebats.html.
[4] MIRACL crypto SDK. https://libraries.docs.miracl.com/.
[5] OpenSSL. https://www.openssl.org/.
[6] Python/C API. https://docs.python.org/2/c-api/index.html.
[7] SageMath. http://www.sagemath.org/.
[8] ZCash. https://z.cash.
[9] libsnark. https://github.com/scipr-lab/libsnark, 2016.
[10] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS*, Oct. 2017.
[11] D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini. A note on high-security general-purpose elliptic curves. *IACR Cryptology ePrint Archive*, 2013. https://eprint.iacr.org/2013/647.
[12] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data. In *IEEE S&P*, May 2015.
[13] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS*, Nov. 2013.
[14] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM CCS*, Nov. 1993.
[15] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO*, Aug. 1990.
[16] E. Ben-Sasson. Concretely efficient computational integrity (CI) from PCPs, 2017. Talk at Workshop on Probabilistically Checkable and Interactive Proofs, June 2017. Slides available at: https://eecs.berkeley.edu/~alexch/docs/pcpip_bensasson.pdf.
[17] E. Ben-Sasson, I. Ben-Tov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, and M. Virza. Computational integrity with a public random string from quasi-linear PCPs. In *EUROCRYPT*, Apr. 2017.
[18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:134, 2017.
[19] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Decentralized anonymous payments from Bitcoin. In *IEEE S&P*, May 2014.
[20] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*, Aug. 2013.
[21] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *IACR TCC*, Oct. 2016.
[22] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, Aug. 2014.
[23] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security*, Aug. 2014.
[24] D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *PKC*, Apr. 2006.
[25] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *IACR TCC*, Mar. 2013.
[26] M. Blum. How to prove a theorem so no one else can claim it. In *ICM*, Aug. 1986.
[27] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, Apr. 2016.
[28] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT*, Dec. 2017.
[29] X. Boyen and B. Waters. Compact group signatures without random oracles. In *EUROCRYPT*, May 2006.
[30] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *PKC*, Apr. 2007.
[31] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *SOSP*, Nov. 2013.
[32] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, Oct. 2017.
[33] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, Aug. 1992.
[34] A. Chiesa, M. A. Forbes, and N. Spooner. A zero knowledge sumcheck and its applications. *CoRR*, abs/1704.02086, 2017.
[35] A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *EUROCRYPT*, Apr. 2015.
[36] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, Jan. 2012.
[37] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *IEEE S&P*, May 2015.

[38] R. Cramer and I. Damgård. Secure signature schemes based on interactive protocols. In *CRYPTO*, Aug. 1995.

[39] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *CRYPTO*, Aug. 1998.

[40] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *IEEE S&P*, May 2016.

[41] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *STOC*, 1991.

[42] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, Aug. 1986.

[43] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *ACM CCS*, Oct. 2016.

[44] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS*, Nov. 2014.

[45] M. Fredrikson and B. Livshits. ZØ: An optimizing distributing zero-knowledge compiler. In *USENIX Security*, Aug. 2014.

[46] R. Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, Aug. 1977.

[47] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013.

[48] I. Giacomelli, J. Madsen, and C. Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security*, Aug. 2016.

[49] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. Cryptology*, 9(3):167–190, 1996.

[50] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[51] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, Aug. 2015. Preliminary version STOC 2008.

[52] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Computing*, 18(1):186–208, 1989.

[53] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT*, Dec. 2006.

[54] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, Aug. 2009.

[55] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.

[56] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, Apr. 2017.

[57] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, Apr. 2008.

[58] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, Apr. 2008.

[59] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Computing*, 28(4):1364–1396, 1999.

[60] S. Hohenberger, S. Myers, R. Pass, and abhi shelat. ANONIZE: A large-scale anonymous survey system. In *IEEE S&P*, May 2014.

[61] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *IEEE CCC*, June 2007.

[62] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, 2007.

[63] Y. T. Kalai and R. Raz. Interactive PCP. In *ICALP*, July 2008.

[64] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, Dec. 2010.

[65] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, May 1992.

[66] J. Kilian. Improved efficient arguments (preliminary version). In *CRYPTO*, pages 311–324, Aug. 1995.

[67] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. TrueSet: Faster verifiable set computations. In *USENIX Security*, Aug. 2014.

[68] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3):143–184, 2003.

[69] H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *IACR TCC*, 2011.

[70] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, Oct. 1992.

[71] U. Maurer. Unifying zero-knowledge proofs of knowledge. In *AFRICACRYPT*, June 2009.

[72] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, Aug. 1987.

[73] S. Micali. Computationally sound proofs. *SIAM J. Computing*, 30(4):1253–1298, 2000.

[74] M. Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[75] A. Naveh and E. Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *IEEE S&P*, May 2016.

[76] A. V. Oppenheim and A. S. Willsky. *Signals and Systems*. Pearson, 1996.

[77] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *IACR TCC*, Mar. 2013.

[78] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE S&P*, May 2013.

[79] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, Aug. 1991.

[80] N. Pippenger. On the evaluation of powers and monomials. *SIAM J. Computing*, 9(2):230–250, 1980.

[81] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, May 1996.

[82] O. Reingold, G. N. Rothblum, and R. D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, June 2016.

[83] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, Oct. 1999.

[84] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[85] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys*, Apr. 2013.

[86] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, Feb. 2012.

[87] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security*, Aug. 2012.

[88] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl/`.

[89] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, May 1997.

[90] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, Aug. 2013. Full version: `https://arxiv.org/abs/1304.3812`.

[91] J. Thaler. A note on the GKR protocol. http://people.seas.harvard.edu/ jthaler/GKRNote.pdf, 2015.

[92] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *USENIX HotCloud Workshop*, June 2012.

[93] K. Turkowski. Filters for common resampling tasks. In *Graphics Gems*, pages 147–165. Academic Press, 1990.

[94] M. Venkitasubramaniam. Personal communication, 2017.

[95] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE S&P*, May 2013.

[96] R. S. Wahby, M. Howald, S. Garg, abhi shelat, and M. Walfish. Verifiable ASICs. In *IEEE S&P*, May 2016.

[97] R. S. Wahby, Y. Ji, A. J. Blumberg, abhi shelat, J. Thaler, M. Walfish, and T. Wies. Full accounting for verifiable outsourcing. In *ACM CCS*, Oct. 2017. Full version: `https://eprint.iacr.org/2017/242/`.

[98] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*, Feb. 2015.

[99] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE S&P*, May 2017.

[100] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. A zero-knowledge version of vSQL. 2017. `https://www.cis.upenn.edu/~danielg3/zkvsql.pdf`.

PEDERSEN COMMITMENT SCHEME

Definitions: Let $\mathcal{G}$ be a (multiplicative) cyclic group of prime order $q_{\mathcal{G}}$ with group operation $\odot$ and inverse $\oslash$. $\mathcal{V}$ publishes generators $g, h \in \mathcal{G}$.

$\mathsf{Com}(m)$**:** $\mathcal{P}$ picks $s \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends $g^m \odot h^s$.

$\mathsf{Open}(\alpha)$**:** $\mathcal{P}$ sends (m, s). $\mathcal{V}$ checks $\alpha \stackrel{?}{=} g^m \odot h^s$.

Vector commitments: For vectors, $\mathcal{V}$ publishes generators $g_1, \ldots, g_n, h \in G$ and $\mathcal{P}$ sends

$$\mathsf{Com}((m_1, \ldots, m_n)) = h^s \odot \bigodot_i g_i^{m_i}$$

FIGURE 4—The Pedersen commitment scheme.

A Instantiations of commitment schemes

In this section, we review the Pedersen commitment scheme [79] (Fig. 4) and related ZK proof protocols. It will be convenient to refer to $g^m \odot h^s$ as $\mathsf{Com}(m; s)$. We refer to s as the *opening* of the commitment $g^m \odot h^s$ to m.

Theorem 7 ([79])**.** *The Pedersen commitment scheme is a non-interactive commitment scheme assuming the hardness of the discrete logarithm problem in $\mathcal{G}$.*

Knowledge of opening. Schnorr [84] shows how $\mathcal{P}$ can give a ZK proof that it knows an x, r such that $C_0 = \mathsf{Com}(x; r)$.

EstablishKnowledgeOfOpening(C_0)

Inputs: $C_0 = g^x \odot h^r$. $\mathcal{P}$ knows x and r.

1. $\mathcal{P}$ picks $t_1, t_2 \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends $\alpha \leftarrow g^{t_1} \odot h^{t_2}$.
2. $\mathcal{V}$ sends a challenge $c \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$.
3. $\mathcal{P}$ sends $z_1 \leftarrow xc + t_1$ and $z_2 \leftarrow rc + t_2$.
4. $\mathcal{V}$ checks that $g^{z_1} \odot h^{z_2} \stackrel{?}{=} C_0^c \odot \alpha$.

Theorem 8 ([84])**.** *EstablishKnowledgeOfOpening is complete, special sound, and honest-verifier perfect zero knowledge.*

Commitment to the same value. Using similar ideas, $\mathcal{P}$ can show in ZK that $C_1 = \mathsf{Com}(v_1; s_1)$ and $C_2 = \mathsf{Com}(v_2; s_2)$ are commitments to the same value, i.e., $v_1 = v_2$. Given $C_u = \mathsf{Com}(u; s_u)$ and a value v, $\mathcal{P}$ can also convince $\mathcal{V}$ that $u = v$.

EstablishOpeningToSameValue(C_1, C_2)

Inputs: $C_1 = g^{v_1} \odot h^{s_1}$ and $C_2 = g^{v_2} \odot h^{s_2}$.
$\mathcal{P}$ knows $v_1 = v_2$, s_1, and s_2.

1. $\mathcal{P}$ picks $r \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends $\alpha \leftarrow h^r$.
2. $\mathcal{V}$ sends a challenge $c \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$.
3. $\mathcal{P}$ sends $z \leftarrow c \cdot (s_1 - s_2) + r$
4. $\mathcal{V}$ checks that $h^z \stackrel{?}{=} (c_1 \oslash c_2)^c \odot \alpha$.

Theorem 9 (Folklore)**.** *EstablishOpeningToSameValue is complete, special sound, and honest-verifier perfect zero knowledge.*

A.1 Proving a product relationship

Figure 5 gives a zero-knowledge proof of knowledge in which $\mathcal{P}$ convinces $\mathcal{V}$ that it has openings to three Pedersen commitments having a product relationship. This is folklore; for example, we know that Maurer [71] describes a very similar protocol.

EstablishProductRelationship(X, Y, Z)

Inputs: $X = g^x \odot h^{r_X}$, $Y = g^y \odot h^{r_Y}$, and $Z = g^{x \cdot y} \odot h^{r_Z}$.
$\mathcal{P}$ knows x,y,r_X,r_Y, and r_Z.

1. $\mathcal{P}$ picks $b_1, \ldots, b_5 \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends

$$\alpha \leftarrow g^{b_1} \odot h^{b_2} \quad \beta \leftarrow g^{b_3} \odot h^{b_4} \quad \delta \leftarrow X^{b_3} \odot h^{b_5}$$

2. $\mathcal{V}$ sends a challenge $c \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$
3. $\mathcal{P}$ sends

$$z_1 \leftarrow b_1 + c \cdot x \qquad z_2 \leftarrow b_2 + c \cdot r_x \qquad z_3 \leftarrow b_3 + c \cdot y$$
$$z_4 \leftarrow b_4 + c \cdot r_y \qquad z_5 \leftarrow b_5 + c(r_z - r_x y)$$

4. $\mathcal{V}$ checks that

$$\alpha \odot X^c \stackrel{?}{=} g^{z_1} \odot h^{z_2} \tag{7}$$
$$\beta \odot Y^c \stackrel{?}{=} g^{z_3} \odot h^{z_4} \tag{8}$$
$$\delta \odot Z^c \stackrel{?}{=} X^{z_3} \odot h^{z_5} \tag{9}$$

FIGURE 5—ZK proof of knowledge for a product relationship (§A.1).

Theorem 10. *Given commitments X, Y, and Z, EstablishProductRelationship proves that Z is a commitment to the product of the values committed in X and Y. This protocol is complete, special sound, and honest-verifier perfect zero knowledge.*

Proof. **Completeness.** It is easy to check that if the prover sends all values as prescribed, then the first two equations hold. Checking that the third equation holds is a straightforward (if slightly tedious) calculation:

$$\begin{aligned}
\delta \odot Z^c &= X^{b_3} \odot h^{b_5} \odot (g^{xy} \odot h^{r_z})^c = X^{b_3} \odot g^{xyc} \odot h^{b_5 + r_z \cdot c} \\
&= X^{b_3} \odot \left((g^x \odot h^{r_x})^{yc} \odot h^{-(r_x \odot yc)} \right) \odot h^{b_5 + r_z \cdot c} \\
&= X^{b_3} \odot X^{yc} \odot h^{b_5 + c(r_z - r_x y)} \\
&= X^{z_3} \odot h^{z_5}.
\end{aligned}$$

Special soundness. To show that the protocol is special sound, for a given theorem statement (X, Y, Z), let (α, β, δ) be a first message, and let $(c, z_1, \ldots, z_5)$ and $(c', z_1', \ldots, z_5')$ be two transcripts such that the verification equations above hold, $c \neq c'$.

Define the variables

$$x = \frac{z_1 - z_1'}{c - c'} \qquad r_x = \frac{z_2 - z_2'}{c - c'}$$
$$y = \frac{z_3 - z_3'}{c - c'} \qquad r_y = \frac{z_4 - z_4'}{c - c'} \qquad w = \frac{z_5 - z_5'}{c - c'}$$

We first show that the values (x, r_x), (y, r_y), are proper openings of the commitments to X and Y.

Recall the following two equations hold:

$$\alpha \odot X^c = g^{z_1} \odot h^{z_2} \qquad \alpha \odot X^{c'} = g^{z_1'} \odot h^{z_2'}$$

It follows by dividing the two equations, that

$$X^{c-c'} = g^{z_1-z_1'} \odot h^{z_2-z_2'}$$

and moreover that

$$X = g^{(z_1-z_1')/(c-c')} \odot h^{(z_2-z_2')/(c-c')} = g^x \odot h^{r_x} \tag{10}$$

which shows that (x, r_x) is indeed an opening for commitment X. The same follows for Y. Finally, since

$$\delta \odot Z^c = X^{z_3} \odot h^{z_5} \qquad \delta \odot Z^{c'} = X^{z_3'} \odot h^{z_5'}$$

it follows again by dividing that

$$Z = X^{(z_3-z_3')/(c-c')} \odot h^{(z_5-z_5')/(c-c')} = X^y \odot h^w$$

Substituting from (10), we have

$$Z = (g^x \odot h^{r_x})^y \odot h^w = g^{xy} \odot h^{r_x y + w}$$

which shows that Z can be opened as the product $(xy, r_x y + w)$.

Perfect ZK. Here, we show an honest-verifier simulation. Using Fiat-Shamir [42] or Cramer-Damgård [38] techniques, one can compile the protocol into a malicious-verifier ZK protocol.

- The simulator $S(X, Y, Z, c)$, on input X, Y, Z and a random challenge c does the following:
  Sample $z_1, \ldots, z_5 \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$, then compute:

$$\alpha \leftarrow g^{z_1} \odot h^{z_2} \oslash X^c \qquad \beta \leftarrow g^{z_3} \odot h^{z_4} \oslash Y^c$$
$$\delta \leftarrow X^{z_3} \odot h^{z_5} \oslash Z^c$$

- Then the simulator outputs the transcript

$$(\alpha, \beta, \delta), c, z_1, z_2, z_3, z_4, z_5$$

By inspection, the transcript satisfies equations (7)–(9) above.

It remains to show that the distribution over transcripts produced by S and those produced by the honest prover and honest verifier are identical. To show this, we show a one-to-one mapping between every transcript produced by the honest prover and a transcript produced by the Simulator. Fix a theorem statement (X, Y, Z) having the correct product relation, a witness, and a challenge c. The Prover's random coins consist of the 5 values b_1, b_2, b_3, b_4, b_5. Given the fixed values, each 5-tuple uniquely defines (α, β, δ) and the values z_1, z_2, z_3, z_4, z_5. This 5-tuple is chosen uniformly over $\mathbb{Z}_{q_{\mathcal{G}}}^5$. Likewise, the simulator S uses random coins $z_1', z_2', z_3', z_4', z_5'$, again chosen uniformly over the same probability space. When S picks the random coins $z_1 = b_1 + c \cdot x$,..., then S produces exactly the same transcript $(\alpha, \beta, \delta), c, (z_1, z_2, z_3, z_4, z_5)$. Thus, for every possible transcript, both the honest prover and simulator produce that transcript with the same probability. □

A.2 Proving a dot-product relationship

In the zero-knowledge proof of knowledge of Figure 6, $\mathcal{P}$ convinces $\mathcal{V}$ that it has openings to one vector commitment $\xi = \mathsf{Com}(\vec{x}; r_\xi)$ and one scalar commitment $\tau = \mathsf{Com}(y; r_\tau)$

EstablishDotProductRelationship$(\xi, \tau, \vec{a})$

Inputs: Commitments $\xi = \mathsf{Com}(\vec{x}; r_\xi)$, $\tau = \mathsf{Com}(y; r_\tau)$, and a vector $\vec{a}$, where $\vec{x}, \vec{a} \in \mathbb{Z}_{q_{\mathcal{G}}}^n$ and $y = \langle \vec{x}, \vec{a} \rangle \in \mathbb{Z}_{q_{\mathcal{G}}}$.
$\mathcal{P}$ knows $\vec{x}$, r_ξ, y, and r_τ.

1. $\mathcal{P}$ samples the vector $\vec{d} \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}^n$ and the values $r_\beta, r_\delta \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends

$$\delta \leftarrow \mathsf{Com}(\vec{d};\ r_\delta) = h^{r_\delta} \odot \bigodot_i g_i^{d_i} \tag{11}$$

$$\beta \leftarrow \mathsf{Com}(\langle \vec{a}, \vec{d} \rangle; r_\beta) = g^{\langle \vec{a}, \vec{d} \rangle} \odot h^{r_\beta} \tag{12}$$

2. $\mathcal{V}$ sends a challenge $c \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$.
3. $\mathcal{P}$ sends

$$\vec{z} \leftarrow c \cdot \vec{x} + \vec{d}, \quad z_\delta \leftarrow c \cdot r_\xi + r_\delta, \quad z_\beta \leftarrow c \cdot r_\tau + r_\beta$$

4. $\mathcal{V}$ checks that

$$\xi^c \odot \delta \stackrel{?}{=} \mathsf{Com}(\vec{z}; z_\delta) = h^{z_\delta} \odot \bigodot_i g_i^{z_i} \tag{13}$$

$$\tau^c \odot \beta \stackrel{?}{=} \mathsf{Com}(\langle \vec{z}, \vec{a} \rangle; z_\beta) = g^{\langle \vec{z}, \vec{a} \rangle} \odot h^{z_\beta} \tag{14}$$

FIGURE 6—ZK vector dot-product proof (§A.2).

such that, for a supplied vector $\vec{a}$ it holds that $y = \langle \vec{x}, \vec{a} \rangle$. Intuitively, this protocol works because

$$\langle \vec{z}, \vec{a} \rangle = \langle c\vec{x} + \vec{d}, \vec{a} \rangle = c\langle \vec{x}, \vec{a} \rangle + \langle \vec{d}, \vec{a} \rangle = cy + \langle \vec{d}, \vec{a} \rangle$$

The above identity is verified in the exponent in Equation (14).

Theorem 11. *The protocol of Figure 6 and Lemma 3 is complete, special sound, and honest-verifier perfect zero knowledge.*

Proof. **Completeness.** If both the prover and the verifier follow the protocol correctly both checks will succeed because

$$\xi^c \odot \delta = \left(h^{r_\xi} \odot \bigodot_i g_i^{x_i} \right)^c \odot h^{r_\delta} \odot \bigodot_i g_i^{d_i}$$
$$= h^{c \cdot r_\xi + r_\delta} \odot \bigodot_i g_i^{c \cdot x_i + d_i} = h^{z_\delta} \odot \bigodot_i g_i^{z_i}$$

$$\tau^c \odot \beta = (h^{r_\tau} \odot g^y)^c \odot g^{\vec{a} \cdot \vec{d}} \odot h^{r_\beta} = h^{c \cdot r_\tau + r_\beta} \odot g^{cy + \langle \vec{a}, \vec{d} \rangle}$$
$$= h^{z_\beta} \odot g^{c \cdot \langle \vec{x}, \vec{a} \rangle + \langle \vec{d}, \vec{a} \rangle} = h^{z_\beta} \odot g^{\vec{z} \cdot \vec{a}}$$

Special soundness. For a given theorem instance $(\xi, \tau, \vec{a})$, let (δ, β) be a first message, and let $(c, \vec{z}, z_\delta, z_\beta)$ and $(c', \vec{z}', z_\delta', z_\beta')$ be two valid transcripts. Since both transcripts satisfy both of $\mathcal{V}$'s checks, we have

$$\xi^c \odot \delta = h^{z_\delta} \odot \bigodot_i g_i^{z_i}$$

$$\xi^{c'} \odot \delta = h^{z_\delta'} \odot \bigodot_i g_i^{z_i'}$$

$$\tau^c \odot \beta = h^{z_\beta} \odot g^{\langle \vec{z}, \vec{a} \rangle}$$

$$\tau^{c'} \odot \beta = h^{z_\beta'} \odot g^{\langle \vec{z}', \vec{a} \rangle}$$

Dividing the top by the bottom in each pair yields

$$\xi = h^{(z_\delta - z'_\delta)/(c-c')} \odot \bigodot_i g_i^{(z_i - z'_i)/(c-c')}$$

$$\tau = h^{z_\beta - z'_\beta/(c-c')} \odot g^{(\vec{z}-\vec{z}')\cdot\vec{a}/(c-c')}$$

which implies that $\vec{x} = (\vec{z}-\vec{z'})/(c-c')$, that $r_\xi = (z_\delta - z'_\delta)/(c-c')$, and that $r_\tau = (z_\beta - z'_\beta)/(c-c')$.

Honest-verifier perfect zero knowledge. (Analogous to the ZK proof for protocol for proving product of commitment.) The zero-knowledge property follows from standard reverse-ordering techniques. In particular, the simulator first picks c, then $\vec{z}, z_\delta, z_\beta$, and finally computes an appropriate first message δ, β which satisfies check equations (13) and (14).

The simulator S, on input the theorem instance $(\xi, \tau, \vec{a})$ and a challenge c does the following:

- Sample $\vec{z'} \xleftarrow{\$} \mathbb{Z}_{q_\mathcal{G}}^n$ and $z'_\delta, z'_\beta \xleftarrow{\$} \mathbb{Z}_{q_\mathcal{G}}$
- Produce values

$$\delta \leftarrow \left(h^{z'_\delta} \odot \bigodot_i g_i^{z'_i} \right) \oslash \xi^c \tag{15}$$

$$\beta \leftarrow \left(g^{\langle \vec{z'}, \vec{a} \rangle} \odot h^{z'_\beta} \right) \oslash \tau^c \tag{16}$$

- Output the transcript $(\delta, \beta), c, (\vec{z}, z_\delta, z_\beta)$.

By inspection, one can certify that the transcript passes both checks of the verifier and the verifier will accept it.

It remains to show that the distribution over transcripts produced by S and those produced by the honest verifier and honest prover are identical. To show this, we show a one to one mapping between every transcript produced by the honest prover and a trancript produced by the simulator. We fix a theorem statement $(\xi, \tau, \vec{a})$, a witness $w = (\vec{x}, r_\xi, r_\tau)$ and a challenge c. The prover's random coins consist of the $(n+2)$-tuple $(\vec{d}, r_\delta, r_\beta)$. Given the fixed statement, witness, and challenge, each $(n+2)$-tuple uniquely determines (δ, β) and the values $\vec{z}, z_\delta, z_\beta$. This $(n+2)$-tuple is chosen uniformly over $\mathbb{Z}_{q_\mathcal{G}}^{n+2}$.

Likewise, the simulator S uses an $(n+2)$-tuple $\vec{z'}, z'_\delta, z'_\beta$ chosen uniformly at random over the probability space. We show a one-to-one mapping between the prover's coins and the simulator's output. In particular, when

$$M(\vec{z'}) = \vec{z'} - c \cdot \vec{x} \quad M(z'_\delta) = z'_\delta - c \cdot r_\xi \quad M(z'_\beta) = z'_\beta - c \cdot r_\tau$$

By inspection, this mapping is one-to-one, and when the prover runs with coins $M(\vec{z'}, z'_\delta, z'_\beta)$, it produces the same transcript as the simulator. Thus, the output distribution of S is identical to that of the prover on this instance. This property holds for all instances, and any challenge c, which concludes the proof. □

A.3 Proof of Lemma 5

Proof. **Completeness.** If the theorem statement

$$\left(C_0, (\alpha_1, \ldots \alpha_n), (M_1, \ldots, M_{n+1}), X, Y, Z\right)$$

holds, the prover knows openings of the commitments $C_0, (\alpha_1, \ldots \alpha_n), X, Y, Z$ and he sends all values as prescribed, it follows from the correctness of EstablishProductRelationship() that X, Y, Z will pass all the checks of that subprocess. Moreover, for all k it holds that

$$\begin{aligned}
&\mathsf{Com}((z_{c_{3,k}}, z_{c_{2,k}}, z_{c_{1,k}}, z_{c_{0,k}}); z_{\delta_k}) \\
&\quad = \mathsf{Com}((c \cdot c_{3,k} + d_{c_{3,k}}, c \cdot c_{2,k} + d_{c_{2,k}}, \\
&\qquad\qquad c \cdot c_{1,k} + d_{c_{1,k}}, c \cdot c_{0,k} + d_{c_{0,k}}); c \cdot r_{\alpha_k} + r_{\delta_k}) \\
&\quad = \left(\mathsf{Com}((c_{3,k}, c_{2,k}, c_{1,k}, c_{0,k}); r_{\alpha_k})\right)^c \odot \\
&\qquad \mathsf{Com}((d_{c_{3,k}}, d_{c_{2,k}}, d_{c_{1,k}}, d_{c_{0,k}}); r_{\delta_k}) \\
&\quad = \alpha_k^c \odot \delta_k
\end{aligned}$$

It also holds that

$$\begin{aligned}
&(C_0^{\rho_1} \odot X^{-J_X} \odot Y^{-J_Z} \odot Z^{-J_Z})^c \odot C \\
&\quad = \mathsf{Com}(c(\rho_1 \cdot s_0 - J_X \cdot x - J_Y \cdot y - J_Z \cdot z) + \langle \vec{J}^*, \vec{d} \rangle; \\
&\qquad\qquad c(\rho_1 \cdot r_{C_0} - J_x \cdot r_X - J_Y \cdot r_Y - J_Z \cdot r_Z) + r_C) \\
&\quad = \mathsf{Com}(c \cdot \langle \vec{J}^*, \vec{\pi}^* \rangle + \langle \vec{J}^*, d \rangle; z_C) \\
&\quad = \mathsf{Com}(\langle \vec{J}^*, \vec{z} \rangle; z_C)
\end{aligned}$$

Generalized special soundness. For theorem instance $(C_0, (\alpha_1, \ldots \alpha_n), (M_1, \ldots, M_{n+1}), X, Y, Z)$, the transcripts:

$$\begin{aligned}
&\left(\mathrm{Tr}_{\mathrm{prod}}, (\delta_1, \ldots, \delta_n), (\rho_1, \ldots \rho_{n+1}), C, c, (\vec{z}, \{z_{\delta_k}\}, z_C)\right) \\
&\left(\mathrm{Tr}_{\mathrm{prod}}, (\delta_1, \ldots, \delta_n), (\rho_1, \ldots \rho_{n+1}), C, c', (\vec{z'}, \{z'_{\delta_k}\}, z'_C)\right) \\
&\left(\mathrm{Tr}'_{\mathrm{prod}}, \ldots\right)
\end{aligned}$$

are sufficient to extract a witness for the statement except with negligible probability, where $\mathrm{Tr}_{\mathrm{prod}}$ and $\mathrm{Tr}'_{\mathrm{prod}}$ are transcripts for EstablishProductRelationship. In this context, by witness we mean openings to the prover's messages that satisfy the checks that the verifier of Gir^{++} does during the corresponding invocation of the sum-check protocol.

The extractor proceeds as follows:

1. Exploiting the first condition checked by the verifier in Figure 1 (Step 7), extract openings for $\{\alpha_k\}$ via

$$\begin{aligned}
\alpha_k^{c-c'} = \mathsf{Com}\big((&z_{c_{3,k}} - z'_{c_{3,k}}, z_{c_{2,k}} - z'_{c_{2,k}}, \\
&z_{c_{1,k}} - z'_{c_{1,k}}, z_{c_{0,k}} - z'_{c_{0,k}}); z_{\delta_k} - z_{\delta'_k}\big)
\end{aligned}$$

   i.e., $r_{\alpha_k} = \frac{z_{\delta_k} - z_{\delta'_k}}{c-c'}$ and $c_{j,k} = \frac{z_{c_{j,k}} - z'_{c_{j,k}}}{c-c'}$, $j \in \{0,1,2,3\}$.
2. Use the extractor for EstablishProductRelationship() and $\mathrm{Tr}_{\mathrm{prod}}, \mathrm{Tr}'_{\mathrm{prod}}$ to extract openings $\hat{x}, r_X, \hat{y}, r_Y, \hat{z}, r_Z$ of X, Y, Z.
3. Use the openings from the previous step to extract an opening for C_0. Specifically, exploiting the second condition checked by the verifier in Figure 1 (Step 7), we have:

$$(c - c')(s_0 - J_X \cdot \hat{x} - J_Y \cdot \hat{y} - J_Z \cdot \hat{z}) = \langle \vec{J}^*, \vec{z} - \vec{z}' \rangle,$$

   from where we can solve for s_0.
4. Check that Equality (5) holds for the extracted values. If not, reject and output $\perp$.

Note that the extractor aborts only when for the extracted witness, $\langle(\sum \rho_k \cdot M_k), \vec{\pi}\rangle = \rho_1 \cdot s_0$ but Equality (5) does not hold. From Lemma 4 the probability that this happens is at most $1/|\mathbb{F}|$, which is negligible when the field is of superpolynomial size. Also, note that we are exploiting the fact that EstablishProductRelationship() ensures that $\hat{z} = \hat{x} \cdot \hat{y}$, since the verifier's checks in the sum-check protocol require this.

Honest-verifier perfect zero knowledge. The simulator, on input statement $(C_0, (\alpha_1, \ldots \alpha_n), (M_1, \ldots, M_{n+1}), X, Y, Z)$ and messages from $\mathcal{V}$, will work as follows. First it uses the simulator for EstablishProductRelationship() to produce a valid transcript Tr_{prod} for EstablishProductRelationship(X, Y, Z). Then, it follows the below steps to simulate the rest of the interaction between $\mathcal{P}$ and $\mathcal{V}$:

1. Pick random $\vec{z}$ and values $z_{\delta_1}, \ldots, z_{\delta_n}, z_C$.
2. For all k, set $\delta_k = \mathsf{Com}((z_{c_{3,k}}, z_{c_{2,k}}, z_{c_{1,k}}, z_{c_{0,k}}); z_{\delta_k}) \oslash \alpha_k^c$ and set $C = \mathsf{Com}(\langle J^*, \vec{z}\rangle; z_C) \oslash (C_0^{\rho_1} \odot X^{-J_X} \odot Y^{-J_Z} \odot Z^{-J_Z})^c$.
3. Output the transcript:

$$\left(\text{Tr}_{\text{prod}}, (\delta_1, \ldots, \delta_n), (\rho_1, \ldots \rho_{n+1}), C, c, (\vec{z}, \{z_{\delta_i}\}, z_C)\right)$$

By construction this passes $\mathcal{V}$'s checks. It remains to show that the distribution of transcripts produced by S and those produced by the honest prover and honest verifier are identical. To show this, we show a one-to-one mapping between every transcript produced by the honest prover and a transcript produced by the simulator. We fix a theorem statement, a witness, and $\mathcal{V}$'s challenges. Since Tr_{prod} is produced by the simulator for EstablishProductRelationship(), its distribution is identical to the distribution of the messages of an honest prover.

Now, we analyze the rest of the transcript. $\mathcal{P}$'s random coins comprise the $(5b_N + 8b_G + 1)$-tuple $(\vec{d}, r_{\delta_1}, \ldots, r_{\delta_n}, r_C)$. Given the fixed statement, witness, and challenges, each such tuple uniquely determines $(\delta_1, \ldots, \delta_n)$, C and the values $\vec{z}, z_{\delta_1}, \ldots, z_{\delta_n}, z_C$. This $(5b_N + 8b_G + 1)$-tuple is chosen uniformly over $\mathbb{Z}_{q_{\mathcal{G}}}^{5b_N+8b_G+1}$.

Likewise, the simulator S uses a $(5b_N + 8b_G + 1)$-tuple $(\vec{z}, z_{\delta_1}, \ldots, z_{\delta_n}, z_C)$ chosen uniformly at random over the probability space. We show a one-to-one mapping between the prover's coins and the simulator's output. We define this mapping as:

$$\begin{aligned} M(\vec{z}) &= \vec{z} - c \cdot \pi^* \\ M(z_{\delta_k}) &= z_{\delta_k} - c \cdot r_{\alpha_k}, \quad k \in \{1, \ldots, n\} \\ M(z_C) &= z_C - c \cdot (\rho_1 \cdot r_{C_0} - J_X r_X - J_Y r_Y - J_Z r_Z) \end{aligned}$$

By inspection, this mapping is one-to-one, and when the prover runs with coins $\left(M(\vec{z}), M(z_{\delta_1}), \ldots, M(z_{\delta_n}), M(z_C)\right)$, it produces the same transcript as the simulator. Thus, the output distribution of S is identical to that of the prover on this instance. This holds for all instances and any challenge c, which concludes the proof. □

A.4 Perfect generalized special soundness for squashed witness (Proof of Lemma 6)

When the vector-product protocol is used to squash the witness, we require some more analysis to argue that a witness can be extracted. The protocol first compresses the witness into one vector commitment T', and then uses the protocol of Appendix A.2 to show the dot-product relation for this vector. Since that protocol is special sound, two transcripts with the same first message can be used to extract the underlying witness; in this case, that witness is the vector:

$$\left(\sum_{i=0}^{\ell_2-1} w_i \check{\chi}_i, \sum_{i=0}^{\ell_2-1} w_{i+\ell_2} \check{\chi}_i, \ \ldots, \ \sum_{i=0}^{\ell_2-1} w_{|w|-\ell_2+i} \check{\chi}_i\right)$$

In other words, each term is a weighted sum of the witness elements, with weights given by the $\check{\chi}_i$ values. Thus, given $\sqrt{|w|}$ accepting transcripts whose $(\check{\chi}_0, \ldots, \check{\chi}_{\ell_2-1})$ are linearly independent (which happens except with negligible probability for randomly selected r_k), an extractor can solve the resulting system of linear equations for the values $w_1, \ldots, w_{|w|}$.

Witness-extended emulation from perfect generalized special soundness. We employ a lemma from Bootle et al. [27, Lem. 1] which shows that generalized special soundness implies witness-extended emulation. Bootle et al. analyze the case where the extractor always succeeds, but their argument establishes the statement below directly.

Lemma 12 (Forking lemma [27])**.** *Let (P, V) be a $(2\mu+1)$-move, public coin interactive protocol. Let E be a witness extraction algorithm that extracts a witness from an $(n_1, \ldots, n_\mu)$-tree of accepting transcripts in probabilistic polynomial time with at most negligible failure probability. Assume that $\prod_{i=\mu}^{n} n_i$ is bounded above by a polynomial in the security parameter λ. Then (P, V) has witness-extended emulation.*

A tree of accepting transcripts is defined as in Section 3.1. The standard notion of special soundness corresponds to $\mu = 1$ and $n = 2$. In Lemma 6, $\mu = 1$, and $n = \sqrt{|w|}$.

B Proofs of Hyrax's security properties

Theorem 13. *The Hyrax protocol is complete: if the prover knows a witness w s.t. $y = C(x, w)$ and follows the prescribed steps in the pseudocode (Appx. D), the verifier will accept.*

Proof. The completeness of Hyrax follows from the completeness of Gir^{++} and the completeness of all the ZK sub-protocols used to establish that the aforementioned checks hold. □

Theorem 14. *The Hyrax Protocol is honest-verifier perfect ZK.*

Proof. On input a theorem instance (C, x, y), and a set of messages $(q'_0, q_{0,0}, q'_1, q_{1,0}, q_{1,1}, \ldots)$ of the verifier, the simulator S proceeds as follows:

1. For $i = 1, \ldots, \sqrt{|w|}$, set $T_i = \mathsf{Com}(0; r_{T_i})$ for random r_{T_i}.
2. Compute $\tilde{V}_y(q'_0, q_{0,0})$ and set $a_0 = \mathsf{Com}(\mathcal{V}_y(q'_0, q_{0,0}); 0)$
3. For each layer $i = 1, \ldots, d$, for all sum-check rounds j, pick $r_{i,j}$ uniformly at random and set $\alpha_{i,j} = \mathsf{Com}((0, 0, 0, 0); r_{i,j})$. At the end of the sum-check, pick $r_{X_i}, r_{Y_i}, r_{Z_i}$ uniformly at random from $\mathbb{F}$ and set $X_i = \mathsf{Com}(0; r_{X_i})$, $Y_i = \mathsf{Com}(0; r_{Y_i})$ and $Z_i = \mathsf{Com}(0; r_{Z_i})$. Use the simulator of EstablishSumCheckRelation() to get a valid transcript Tr_i for the theorem statement $\left(a_{i-1}, (\alpha_{i,1}, \ldots \alpha_{i,n}), (M_1, \ldots, M_{n+1}), X_i, Y_i, Z_i\right)$. Pick random $\mu_{i,0}, \mu_{i,1}$ from $\mathbb{F}$ and set $a_i = X_i^{\mu_{i,0}} \odot Y_i^{\mu_{i,1}}$ (cf. Line 19 of Figure 7).

4. For all $j = 0, \ldots, b_G$ pick r_{H_j} uniformly at random from $\mathbb{F}$ and set $\mathsf{Com}(H_j) = \mathsf{Com}(0; r_{H_j})$
5. Run the simulator of EstablishKnowledgeOfOpening() for the values $\mathsf{Com}(H_0), \ldots, \mathsf{Com}(H_{b_G})$ and get back the accepting transcripts $\mathrm{Tr}_{H_0}, \ldots, \mathrm{Tr}_{H_{b_G}}$
6. Run the simulator of EstablishOpeningToTheSameValue() for the input statement $(\mathsf{Com}(H_0), X)$ and get back the accepting transcript $\mathrm{Tr}_{\mathrm{same},X}$
7. Run the simulator of EstablishOpeningToTheSameValue() for the input statement $(\mathsf{Com}(H_{b_G}) \odot \ldots \odot \mathsf{Com}(H_0), Y)$ and get back the accepting transcript $\mathrm{Tr}_{\mathrm{same},Y}$
8. Compute q_d, ζ and T' deterministically as in Lines 24–26 of Figure 8
9. Run the simulator of EstablishDotProductRelationship() for the input statement $(T', \zeta \oslash g^{(1-q_d[0])\vec{V}_x(q_d[1,\ldots,b_N+b_G-1])}, R)$ and get back the accepting transcript $\mathrm{Tr}_{\mathrm{dotProd}}$
10. Output the transcript:
$(T_1, \ldots, T_{\sqrt{|w|}}, q'_0, q_{0,0}, q_{1,0}, \alpha_{1,0}, \ldots, X_1, Y_1, Z_1, \mathrm{Tr}_1, \ldots,$
$\mathrm{Tr}_d, \mathsf{Com}(H_0), \ldots, \mathrm{Tr}_{H_{b_G}}, \mathrm{Tr}_{\mathrm{same},X}, \mathrm{Tr}_{\mathrm{same},Y}, \tau, \mathrm{Tr}_{\mathrm{dotProd}})$

The above transcript is accepting: The verifier can only reject during one of the ZK subroutines ProveDotProductRelationship(), ProveKnowledgeOfOpening(), etc., but the simulators of these routines produce accepting transcripts for each of the input theorem statements.

By a standard hybrid argument, the distribution of transcripts produced by S and those produced by the honest prover and honest verifier are identical. We fix a theorem statement (C, x, y), a witness w and verifier's challenges $q'_0, q_{0,0}, \ldots$. Recall $\mathsf{View}(\langle \mathcal{P}(w), \mathcal{V}^*(z)\rangle(x))$ denotes the distribution of transcripts that the prover produces when interacting with the honest verifier. In the following we define experiments $\mathsf{Exp}_1, \mathsf{Exp}_2, \ldots$ which produce a distribution of transcripts, and show that each is identical to the previous.

1. Let Exp_1 be the experiment that outputs a transcript in which the prover behaves as the honest prover except that in Line 28 of Figure 8, instead of executing the protocol EstablishDotProductRelationship honestly, the protocol's simulator is used to produce the transcript for the same theorem. From Theorem 11, since this simulator produces transcripts that are identically distributed to those that an honest prover produces, the distributions Exp_1 and $\mathsf{View}(\langle \mathcal{P}(w), \mathcal{V}^*(z)\rangle(x))$ are identical.
2. Let Exp_2 be the experiment in which the prover behaves as the prover in Exp_1 except that in Line 3 of Figure 8, a commitment to 0 is used. By Theorem 7 which establishes that the Pedersen commitment scheme is perfectly hiding, each of these commitments to zero is identically distributed to commitment used in Exp_1. As a result, the distributions Exp_2 and Exp_1 are identically distributed.
3. Let Exp_3 be the experiment in which the prover behaves as the prover in Exp_2 except that in each of the Lines 21 and 22 of Figure 8, the simulator of the protocol EstablishOpeningToSameValue is used to generate the transcript instead of executing the protocol. By Theorem 9 this simulator produces transcripts for each of these relations that are identical to those that an honest prover would produce.
4. Let Exp_4 be the experiment in which the prover behaves as the prover in Exp_3 except that in the b_G iterations of Line 20 of Figure 8, the simulator of the protocol EstablishKnowledgeOfOpening is used to generate a transcript instead of executing it honestly. From Theorem 8, this simulator produces transcripts that are identically distributed to those that an honest prover produces. So, the distributions H_4 and H_3 are identical.
5. Let Exp_5 be the experiment in which the prover behaves as the prover in Exp_4 except that in Line 18 of Figure 8, commitments to 0 are used. By Theorem 7 which establishes that the Pedersen commitment scheme is perfectly hiding, the distribution Exp_5 is identical to the distribution Exp_4
6. For all sum-check iterations $i = d, \ldots, 1$:
   - Let $\mathsf{Exp}_{3(d-i)+6}$ be the experiment in which the prover behaves as the prover of the previous experiment $\mathsf{Exp}_{3(d-i)+5}$ except that in Line 54 of figure 9, the simulator for the protocol EstablishSumCheckRelation is used to generate the transcript. From Lemma 5 this simulator produces transcripts that are identically distributed to those that an honest prover produces. So, the distributions $\mathsf{Exp}_{3(d-i)+6}$ and $\mathsf{Exp}_{3(d-i)+5}$ are identical.
   - Let $\mathsf{Exp}_{3(d-i)+7}$ be the experiment in which the prover behaves as the prover of the previous experiment $Exp_{3(d-i)+6}$ except that in Line 52 of figure 9 produces the commitments X_i, Y_i, Z_i by committing to 0. By Theorem 7 which establishes that the Pedersen commitment scheme is perfectly hiding, the distribution $\mathsf{Exp}_{3(d-i)+7}$ is identical to the distribution $\mathsf{Exp}_{3(d-i)+7}$.
   - Let $\mathsf{Exp}_{3(d-i)+8}$ be the experiment in which the prover behaves as the prover of the previous experiment $\mathsf{Exp}_{3(d-i)+8}$ except that for all SumCheck rounds $j = 1 = n, \ldots, 1$, in Lines 19 and 47 of figure 9 the commitments α_i are produced by committing to the zero vector. By Theorem 7 which establishes that the Pedersen commitment scheme is perfectly hiding, the distribution $\mathsf{Exp}_{3(d-i)+8}$ is identical to the distribution $\mathsf{Exp}_{3(d-i)+7}$

Thus, $\mathsf{View}(\langle \mathcal{P}(w), \mathcal{V}^*(z)\rangle(x))$ is identically distributed to $\mathsf{Exp}_{3(d-1)+8}$. The experiment $\mathsf{Exp}_{3(d-1)+8}$ is identical to running the Simulator S on the instance (C, x, y) and verifier's challenges $q'_0, q_{0,0}, \ldots$. As a result, $\mathsf{Exp}_{3(d-1)+8}$ corresponds to the distribution of transcripts that S produces which completes the theorem. □

Theorem 15. *Under the discrete logarithm assumption, Hyrax-interactive satisfies witness extended emulation.*

Proof. We construct an extractor E that simultaneously outputs a transcript and a witness for a given theorem x with roughly the same probability that prover $\mathcal{P}^*$ causes the honest verifier to accept theorem (C, x, y). At a high level, this extractor E uses $\mathcal{P}^*$ as an oracle, and runs the extractor from Lemma 6 in

order to find a witness w. It checks the validity of the witness by verifying that $C(x, w) = y$ and outputs w or else aborts by outputting $\perp$. We show that when the extractor succeeds in recovering w, then (except with negligible probability) it indeed outputs w and thus satisfies witness-extended emulation. The last condition follows by the soundness of the Gir^{++} protocol: a $\mathcal{P}^*$ that succeeds at convincing the verifier to accept with some non-negligible probability ϵ, but which causes E to abort, can be used to break the soundness of Gir^{++}. We use a proof technique borrowed from the elegant analysis of various parallel repetition theorems where the same style of soundness reduction (albeit more complicated) is required.

To simplify the presentation, we assume that the Hyrax-interactive verifier runs all zero-knowledge proofs at the very end of the protocol (even though the pseudocode describes these checks as occurring over the course of the interaction); we refer to this ZK challenge step as Step $(*)$. By moving all sigma protocols to the end in Hyrax, and by standard AND composition of sigma protocols [33], all ZK proofs can use the same verifier challenge $c \in \mathbb{F}$ and the combined proof enjoys the same honest-verifier zero knowledge and special soundness properties (we think of Step (*) as consisting of one giant ZK proof instead of many separate ZK proofs). Similarly, the Gir^{++} verifier defers all checks to the end of the interaction. This change to Gir^{++} does not affect soundness.

On input theorem statement (C, x, y), the extractor $E^{\mathcal{P}^*}(C, x, y)$ works as follows:

1. Run the honest Hyrax-interactive verifier $\mathcal{V}$ on instance (C, x, y) with prover $\mathcal{P}^*(C, x, y)$ and record the partial transcript Tr' until Step $(*)$ when the zero-knowledge proof is given.
2. By Lemma 12, there exist witness-extended emulators $E_w^{\mathcal{P}^*}$ and $E_O^{\mathcal{P}^*}$ for the protocols of Lemma 6 and Theorem 8, respectively. Run $(\mathrm{Tr}''_e, w_e) \leftarrow E_w^{\mathcal{P}^*}(C, x, y)$ to extract the witness w_e. Run $(\mathrm{Tr}''_{H_i}, (H_i, r_{H_i})) \leftarrow E_O^{\mathcal{P}^*}$ to extract openings for each $\mathsf{Com}(H_i)$ (Fig. 7, Line 26), and use these openings to compute the opening (m_ζ, r_ζ) of ζ (Fig. 7, Line 32).
3. Run the rest of the ZK sub-protocols of Step (*) and record the transcript $\mathrm{Tr}''_{\mathrm{rest}}$. Let $\mathrm{Tr}'' = \mathrm{Tr}''_e||\mathrm{Tr}''_{H_0}||\mathrm{Tr}''_{H_1}||\ldots||\mathrm{Tr}''_{\mathrm{rest}}$, and let $\mathrm{Tr} = \mathrm{Tr}'||\mathrm{Tr}''$.
4. If m_ζ is not consistent with (x, w_e) or if Tr is rejecting, output $(\mathrm{Tr}, 0)$ and halt. Otherwise, if $C(x, w_e) = y$, then output (Tr, w_e). Otherwise, output $\perp$.

Fix a polynomial-time adversary A and $\mathcal{P}^*$. Without loss of generality, assume that $A, \mathcal{P}^*$ are deterministic (i.e., their best random tapes are hard-wired). Thus, the instance and state given by $(u_\lambda{=}(C, x, y), s_\lambda) \leftarrow A(1^\lambda)$ are just indexed on the security parameter. We must show that E runs in expected polynomial time (which follows by inspection of each step) and that

$$\Pr\left[\mathrm{Tr} \leftarrow \mathrm{tr}\langle \mathcal{P}^*(u_\lambda, s_\lambda), \mathcal{V}(u_\lambda)\rangle : A(\mathrm{Tr}) = 1\right] \tag{17}$$

and

$$\Pr\left[\begin{array}{l}(\mathrm{Tr}, w) \leftarrow E^{\mathcal{P}^*(u_\lambda, s_\lambda)}(u_\lambda) : A(\mathrm{Tr}) = 1 \wedge \\ \text{if Tr is an accepting transcript, } C(x, w) = y.\end{array}\right] \tag{18}$$

differ by a negligible amount in the security parameter.

When E outputs a pair, it outputs the transcript Tr produced by an honest verifier interacting with $\mathcal{P}^*$, and thus the probability that $A(\mathrm{Tr}) = 1$ when E does not abort is identical in (18) and (17). As we show below in Claim 16, the probability that E aborts is a negligible function $\eta(\cdot)$, and thus, it follows that $\Pr\left[(\mathrm{Tr}, w) \leftarrow E^{\mathcal{P}^*(u,s)}(u) : A(\mathrm{Tr}) = 1\right] \geq (1 - \eta(\cdot)) \cdot$ Eq. (17). Furthermore, when E does not abort and the transcript is accepting, then because of Step 4, E outputs a pair (Tr, w) such that $C(x, w) = y$. Thus, the second condition of Eq. (18) also holds, and equations (17) and (18) differ by at most a negligible function, which completes the proof of the theorem.

E only aborts when $\mathcal{P}^*$ succeeds in creating an accepting transcript, extractor E_w succeeds in extracting a witness w_e, but $C(x, w_e) \neq y$. Denote this event as badwit.

Claim 16. *For every $\mathcal{P}^*$, there exists a $\hat{\mathcal{P}}$ that runs in expected time* $\mathrm{poly}(|u_\lambda| + \mathrm{time}(\mathcal{P}^*))$ *such that if* $\Pr[\mathsf{badwit}] = \epsilon(\lambda)$*, then*

$$\Pr[\langle \hat{\mathcal{P}}(u_\lambda, s_\lambda), \mathcal{V}_g(u_\lambda)\rangle = 1] > \epsilon(\lambda)^2 - \eta(\lambda)$$

where $\mathcal{V}_g$ is the Gir^{++} verifier and η is a negligible function.

By the soundness of Gir^{++} from Theorem 1, it follows that

$$\epsilon(\lambda)^2 - \eta(\lambda) < O(|u_\lambda|/|\mathbb{F}|)$$

and by rearranging, that $\epsilon(\lambda) < \sqrt{O(|u_\lambda|/|\mathbb{F}|) + \eta(\lambda)}$ which shows that ϵ is a negligible function because $|\mathbb{F}|$ is assumed exponential in the security parameter.

The idea of the construction of $\hat{\mathcal{P}}$. Both the Gir^{++} verifier's $(\mathcal{V}_g)$ challenge messages and the Hyrax verifier's $(\mathcal{V})$ challenge messages are $(\mu_{0,0}, \mu_{0,1}, r_{1,1}, r_{1,2}, \ldots, r_{d,b_G}, \tau)$, and in addition, the Hyrax verifier makes a single challenge c in Step $(*)$. For simplicity, we denote these challenges $\vec{r} = (r_1, \ldots, r_\ell) \in \mathbb{F}^\ell$ and $c \in \mathbb{F}$ and assume that $|\mathbb{F}| > 2^\lambda$.

The only difference between $\mathcal{V}_g$ and $\mathcal{V}$ is that the latter performs its checks "under the commitments" by verifying a ZK proof instead of directly checking the provers responses $(m_1, \ldots, m_\ell)$ after every step. Thus the job of $\hat{\mathcal{P}}$ is to forward the messages "under the commitments" from $\mathcal{P}^*$ to $\mathcal{V}_g$.

In particular, $\hat{\mathcal{P}}$ needs to sample an accepting transcript between $\mathcal{V}$ and $\mathcal{P}^*$, and then rewind the challenge message of the ZK proof to extract the witness w_e from the session and initiate a session of Gir^{++} with $\mathcal{V}_g$ on the theorem statement $C(x, w_e) = y$. As it receives the challenges from $\mathcal{V}_g$ for the sum-check iterations, it forwards the same challenge to $\mathcal{P}^*$, awaits a response commitment, and then extracts the value from the commitment by completing the execution and extracting a witness from the ZK proof at Step $(*)$ by using the (generalized) special soundness of each sub-protocol along with Lemma 12.

Bounding the probability that $\mathcal{P}^*$ aborts. A natural concern is handling provers $\mathcal{P}^*$ that abort the Hyrax protocol during $\hat{\mathcal{P}}$'s attempts to find a completion of a partial transcript. Intuitively, $\mathcal{P}^*$ succeeds on ϵ-fraction of all $\mathcal{V}$ challenges, and thus sufficient sampling should lead to a success with high probability.

More formally, consider a j-move partial execution between $\mathcal{P}^*$ and $\mathcal{V}$ in which $\mathcal{V}$'s challenges are $\vec{r}_j = (r_1, \ldots, r_j)$. A *continuation* of $\vec{r}_j$ is a set of challenges $(r_{j+1}, \ldots, r_\ell, c)$, and a *good*

continuation is one in which P^* responses to $(r_1, \ldots, r_\ell, c)$ causes V to accept. Define the set G_λ to be $G_\lambda = \{\vec{r} = (r_1, \ldots, r_\ell)\}$ the set of ℓ-move challenges such that each prefix $\vec{r}_j$ of $\vec{r} \in \mathcal{G}_\lambda$, has an ϵ/ℓ-fraction of good continuations. These are the "heavy" challenges for which $\mathcal{P}^*$ produces accepting transcripts that facilitate extraction via special-soundness. Let $\nu_\lambda = \Pr_{\vec{r} \xleftarrow{\$} \mathbb{F}^\ell}[\vec{r} \in G_\lambda]$. Next we lower bound ν_λ.

Lemma 17. *$\nu_\lambda \geq \epsilon/8$ for $\ell \geq 3, \ell \in \mathbb{N}$.*

Proof. Consider the tree of transcripts with leaves $(r_1, \ldots, r_\ell) \in \mathbb{F}^\ell$ representing the portion of an execution of Hyrax right before the challenge c for the ZK proof is given. Level j of this tree contains the internal nodes $\vec{r}_j = (r_1, \ldots, r_j) \in \mathbb{F}^j$ (so there are $|\mathbb{F}|^j$ nodes at layer j of the tree); we are interested in analyzing *good* continuations over these leaves. Let $\rho_{\vec{r}_j}$ be the fraction of good continuations for prefix $\vec{r}_j$. Suppose that over all nodes at level i, an ϵ_i fraction of continuations are good. Define the heavy set $\mathcal{H}_i$ as those prefixes for which at least an ϵ_i/ℓ-fraction of continuations succeed.

Claim 18. *If an ϵ_i fraction of continuations of all nodes $\vec{r}_i$ at level i succeed, then at least an $\epsilon_i(1 - 1/\ell)$ fraction of continuations of nodes in $\mathcal{H}_i$ at level i succeed. That is, if $\sum_{\vec{r}_i \in \mathbb{F}^i} \rho_{\vec{r}_i} > \epsilon_i$, then $\sum_{\mathcal{H}_i} \rho_{\vec{r}_i} \geq \epsilon_i(1 - \frac{1}{\ell-1})$.*

Proof. Partition the sum

$$|\mathbb{F}|^{(\ell-i)} \sum_{\vec{r}_i \in \mathbb{F}^i} \rho_{\vec{r}_i} = |\mathbb{F}|^{(\ell-i)} \left(\sum_{\mathcal{H}_i} \rho_{\vec{r}_i} + \sum_{\overline{\mathcal{H}_i}} \rho_{\vec{r}_i} \right) \geq \epsilon_i \cdot |\mathbb{F}|^\ell \quad (19)$$

Using the fact that $|\mathcal{H}_i| + |\overline{\mathcal{H}_i}| = |\mathbb{F}|^i$, we bound $|\overline{\mathcal{H}_i}|$ by overestimating its weight as follows:

$$\begin{aligned}
|\mathbb{F}|^{(\ell-i)} \left(|\mathcal{H}_i| \cdot 1 + \left|\overline{\mathcal{H}_i}\right| \left(\frac{\epsilon_i}{\ell}\right) \right) &\geq \epsilon_i \cdot |\mathbb{F}|^\ell \\
|\mathbb{F}|^{(\ell-i)} \left((|\mathbb{F}|^i - \left|\overline{\mathcal{H}_i}\right|) + \left|\overline{\mathcal{H}_i}\right| \left(\frac{\epsilon_i}{\ell}\right) \right) &\geq \epsilon_i \cdot |\mathbb{F}|^\ell \\
|\mathbb{F}|^{(\ell-i)} \left(|\mathbb{F}|^i + \left|\overline{\mathcal{H}_i}\right| \left(-1 + \left(\frac{\epsilon_i}{\ell}\right)\right) \right) &\geq \epsilon_i \cdot |\mathbb{F}|^\ell \\
|\mathbb{F}|^{(\ell-i)} \left|\overline{\mathcal{H}_i}\right| \left(-1 + \left(\frac{\epsilon_i}{\ell}\right)\right) &\geq (-1 + \epsilon_i) \cdot |\mathbb{F}|^\ell \\
\left|\overline{\mathcal{H}_i}\right| &\leq \frac{(1 - \epsilon_i)}{\left(1 - \left(\frac{\epsilon_i}{\ell}\right)\right)} \cdot |\mathbb{F}|^i \quad (20)
\end{aligned}$$

Substituting (20) into (19)

$$\begin{aligned}
\sum_{\mathcal{H}_i} \rho_{\vec{r}_i} &\geq \epsilon_i \cdot |\mathbb{F}|^i - \sum_{\overline{\mathcal{H}_i}} \rho_{\vec{r}_i} \\
&\geq \epsilon_i \cdot |\mathbb{F}|^i - \frac{(1 - \epsilon_i)}{\left(1 - \left(\frac{\epsilon_i}{\ell}\right)\right)} \cdot |\mathbb{F}|^i \cdot \frac{\epsilon_i}{\ell} \\
&= \epsilon_i \cdot |\mathbb{F}|^i \left[1 - \frac{(1 - \epsilon_i)}{\ell - \epsilon_i} \right] \\
&\geq \epsilon_i \cdot |\mathbb{F}|^i \left[1 - \frac{1}{\ell - 1} \right]
\end{aligned}$$

□

Thus, at level i, the probability mass over the "heavy" prefixes remains roughly the same. At level ℓ in the tree (i.e., full challenges), we have by assumption that an ϵ fraction succeeds, and thus by the calculation above, at least $(1 - 1/(\ell-1))$ fraction are heavy leaves for which ϵ/ℓ-fraction of the continuations (over the ZK challenge c) succeed. Now consider the parents of these heavy leaves at level $\ell - 1$. At least an $\epsilon_{\ell-1} = \epsilon(1 - 1/(\ell-1))$ fraction of nodes at this level have children which are heavy. Applying Claim 18, it follows that an $\epsilon_{\ell-1} \cdot (1 - 1/(\ell-1))$ fraction of the nodes at this level are heavy. By induction, we have that at the top-level of the tree, at least a fraction of all challenges

$$\epsilon \left[1 - \frac{1}{\ell - 1} \right]^\ell \geq \epsilon/8.$$

are heavy, because $(1 - 1/(x-1))^x \geq 1/8$ for $x \geq 3, x \in \mathbb{N}$. □

We now consider the task of sampling an accepting transcript that allows extracting w starting from a partial transcript with challenges $\vec{r}_j = (r_1, \ldots, r_j)$. Define procedure $\text{SAM}^{\mathcal{P}^*}_{\lambda,u}(\vec{r}_j)$ as:

1. For up to t attempts:
   (a) Sample $(r'_{j+1}, \ldots, r'_\ell) \leftarrow \mathbb{F}^{\ell-j}$
   (b) Run $\mathcal{V}$ with $\mathcal{P}^*$ (from its current state) using challenges $(r'_{j+1}, \ldots, r'_\ell)$ until $\hat{\mathcal{P}}$ generates the first message of the ZK proof, and then sample t transcripts with randomly sampled ZK challenges $c \leftarrow \mathbb{F}$. Succeed if $\sqrt{|u_\lambda|}$ transcripts accept, and return the accepting transcripts.

Since $|u_\lambda| > |w|$, $\sqrt{|u_\lambda|}$ transcripts are sufficient to extract w (Appx. A.4). Further, SAM always runs in time polynomial in time($\mathcal{P}^*$) and $\sqrt{|u_\lambda|}$ (which are both polynomial in the instance size) and t, which we set below. More importantly, for $j \in \{1, \ldots, \ell\}$, conditioned on $\vec{r}_j$ being the prefix of some $\vec{r} \in G_\lambda$, then SAM succeeds with high probability. In particular, for SAM to fail, all t attempts at sampling $(r_{j+1}, \ldots, r_\ell)$ must fail to yield a candidate $\vec{r} \in G_\lambda$ and more than $t - \sqrt{|u_\lambda|}$ attempts to find good continuations of each such candidate fail. For the first case, because the samples are independent, the failure probability is less than $(1 - \epsilon/8)^t$. For the second case, failure occurs when there are fewer than $\sqrt{|u_\lambda|}$ accepting transcripts. Conditioned on the prefix being in G_λ, an ϵ/ℓ fraction of continuations result in accepting transcripts, so sampling $\sqrt{|u_\lambda|}$ accepting transcripts requires an expected $\sqrt{|u_\lambda|} \cdot \ell/\epsilon$ attempts. Setting $t = 10\sqrt{|u_\lambda|} \cdot \ell/\epsilon \cdot \lambda$ makes the first case negligible by inspection and the second case negligible by a Chernoff bound.

Details of $\hat{\mathcal{P}}$. For theorem (C, x, y), $\hat{\mathcal{P}}$ does the following:

1. Sample $\text{Tr}_{\text{full}} \leftarrow \langle \mathcal{P}^*, \mathcal{V}(C, x, y) \rangle$. If Tr_{full} is not accepting, then abort.
2. Rewind $\mathcal{P}^*$ to Step $(*)$ and sample accepting transcripts until there are enough to extract the witness w_e and an opening (m_ζ, r_ζ) of ζ via the perfect generalized special soundness of the ZK step (Step (*)) and Lemma 12. Send w_e to Gir^{++} verifier $\mathcal{V}_g$ (Line 3 of Figure 10), thereby making the claim that $C(x, w_e) = y$.

   Rewind $\mathcal{P}^*$ to after it sends its first commitment message

(Line 3, Fig. 12). From this point, $\hat{\mathcal{P}}$ plays man in the middle between $\mathcal{V}_g$ and $\mathcal{P}^*$ as follows.

3. Repeat to determine each message that the Gir^{++} prover $\hat{\mathcal{P}}$ sends to the Gir^{++} verifier $\mathcal{V}_g$:

(a) For message j, await the random challenge r_j from $\mathcal{V}_g$, forward r_j to $\mathcal{P}^*$. Run SAM to generate enough accepting continuations and ZK transcripts to extract prover messages $(m_1^j, \ldots, m_\ell^j)$ for every committed message in the protocol.

(b) For any $k < j$, if there exists a pair $m_k^k \neq m_k^j$, (i.e., the extracted value of message k differs from a previous extraction of message k), then abort.

(c) Forward the extracted message m_j^j to $\mathcal{V}_g$.

During its execution, $\hat{\mathcal{P}}$ samples fresh random challenges on behalf of the Hyrax verifier, and it receives challenge messages from $\mathcal{V}_g$. Let coll denote the event that two such samples are equal (over the entire execution including all sampling of continuations). Because both $\hat{\mathcal{P}}$ and $\mathcal{V}_g$ choose samples randomly and uniformly from $\mathbb{F}$, and because the expected number of samples chosen is $O(\text{poly}(d \log(NG)))$ over all rewinds, it follows that $\Pr[\text{coll}] \leq \eta_1(\lambda)$ for a negligible function η_1. For the rest of this argument, we condition on the event $\overline{\text{coll}}$, which implies that every extraction of a witness succeeds by perfect generalized special soundness and Lemma 12.

$\hat{\mathcal{P}}$ may abort in Step 3b. Denote this event by bind; when this event occurs, $\hat{\mathcal{P}}$ can be modified to output a commitment α_j and two different openings of α_j. This event occurs with negligible probability η_2 by the binding of the Pedersen commitment, and we also condition the rest of the analysis on the event $\overline{\text{bind}}$.

We can view verifier $\mathcal{V}_g$ as sampling all of its challenges $\vec{r} \in \mathbb{F}^\ell$ at the start of the protocol (but only sending them one by one). Let event good occur when $\mathcal{V}_g$'s sampled challenge $\vec{r} \in G_\lambda$. By Lemma 17, this event occurs with probability $\epsilon/8$. Even conditioned on good, the invocations of SAM may fail, via the union bound, with negligible probability at most $\eta_3 \geq e^{-\lambda} \cdot \text{poly}(d, \log(NG))$.

The first step of $\hat{\mathcal{P}}$ succeeds with probability ϵ. When this occurs, the second step runs an expected $\sqrt{|u_\lambda|}/\epsilon$ times to recover enough transcripts to extract the witness. Conditioned on the four events above, $\hat{\mathcal{P}}$ always succeeds in convincing $\mathcal{V}_g$, thus:

$$\begin{aligned}\Pr[\langle \hat{\mathcal{P}}(u_\lambda, s_\lambda), V_g(u_\lambda)\rangle = 1] &\geq \epsilon \cdot \epsilon/8 \cdot (1 - \eta_1(\lambda)) \\ &\quad \cdot (1 - \eta_2(\lambda)) \cdot (1 - \eta_3(\lambda)) \\ &\geq \epsilon^2/8 - \eta_4(\lambda)\end{aligned}$$

where η_4 is a negligble function and η_1, η_2, and η_3 correspond to the events bind, coll and sam.

To compute the running time, there are three cases: (a) when step 1 fails (with probability $1-\epsilon$), (b) when step 1 succeeds and events good, bind, and coll all occur, (c) when step 1 succeeds and $\overline{\text{good}}$ occurs. In (a), $\hat{\mathcal{P}}$ runs one execution of $\hat{\mathcal{P}}$, in (b), $\hat{\mathcal{P}}$ makes an expected polynomial number of executions of $\hat{\mathcal{P}}$, and in (c), we use the loop bound t to ensure that the runtime remains polynomial.

Finally, the loop bound t in SAM relies on ϵ. We can either estimate this probability after Step 1 using an analysis technique from Goldreich and Kahan [49], or we can be given this value as advice for this security parameter. Furthermore, we can use a standard technique of aborting the reduction after 2^λ steps to catch the unexpected steps that may occur due to sampling errors. □

C Randomized checks "inside the AC"

We describe a simple technique for running randomized tests "inside the AC." We use this technique improve the cost of modulo-2^{32} addition in an AC for SHA-256, but it is applicable to many randomized checks. Zhang et al. use a related approach to verify set intersections in vSQL [99].

Our implementation of SHA-256 uses a prior technique [19] to compute $h = f + g \mod 2^{32}$ in an AC over a large prime field. Specifically, $\mathcal{P}$ supplies witness elements $\{b_i\}$ that are purportedly the binary digits of $f + g$, i.e., $f + g = \sum_i 2^i b_i$. The AC computes and outputs the value $f + g - \sum_i 2^i b_i$, which $\mathcal{V}$ checks is equal to zero. The AC also encodes "bit tests" that ensure $b_i \in \{0, 1\}$, namely, it computes and outputs $\{b_i \cdot (1 - b_i)\}$, which $\mathcal{V}$ also checks are zero. Finally, the AC uses $\{b_i\}$ to compute h, i.e., $h = \sum_{i=0}^{31} 2^i b_i = f + g \mod 2^{32}$.

In our implementation, the SHA-256 AC entails more than 7 000 bit tests. Because Gir^{++} requires layered ACs (§3.2), the results of these tests must be passed layer-by-layer to the output, meaning that they cost $\approx 7000d$ gates in total. As a result, bit tests nearly double the size of the AC.

To avoid this, we observe that it is safe for $\mathcal{V}$ to believe that all bit tests have passed if $\sum_i r_i \cdot b_i \cdot (1 - b_i) = 0$ for $r_i \in_R \mathbb{F}$—as long as $\mathcal{P}$ *does not know* $\{r_i\}$ when it chooses $\{b_i\}$. To ensure this, we introduce a minor modification to Hyrax. Specifically, $\mathcal{V}$ first encodes the above check into the AC. Then, during protocol execution, after $\mathcal{V}$ sends x, $\mathcal{P}$ commits to the witness (including $\{b_i\}$; §6); now $\mathcal{V}$ chooses and sends $\{r_i\}$, after which $\mathcal{P}$ returns y and the protocol continues as normal. $\mathcal{V}$ rejects if the output corresponding to the above sum is not 0. Informally, since $\mathcal{P}$ is forced to choose its $\{b_i\}$ before $\mathcal{V}$ chooses $\{r_i\}$, it cannot "fool" the test.

Completeness follows by inspection; soundness follows from the Schwartz-Zippel lemma. This protocol is public coin, so it can be made non-interactive with the Fiat-Shamir heuristic. Finally, we note that in the data-parallel setting $\mathcal{V}$ can supply a single $\{r_i\}$ for all N sub-ACs (because $\mathcal{P}$ commits to the entire witness at once).

D Hyrax-Interactive pseudocode

In this section, we provide pseudocode for Hyrax-Interactive. Figure 7 details $\mathcal{V}$'s work; Figures 8 and 9 detail $\mathcal{P}$'s.

E Gir^{++} specification

In this section, we provide pseudocode for Gir^{++}. Figures 10 and 11 detail $\mathcal{V}$'s work; Figures 12 and 13 detail $\mathcal{P}$'s.

1: **function** HYRAX-VERIFY(ArithCircuit c, input x, output y)
2: // Receive commitments to the rows of the matrix T
3: $(T_1, \ldots, T_{\sqrt{|w|}}) \leftarrow$ ReceiveFromProver() // see Line 3 of Figure 8
4: $b_N \leftarrow \log N, b_G \leftarrow \log G$
5: $(q'_0, q_{0,0}) \xleftarrow{R} \mathbb{F}^{b_N} \times \mathbb{F}^{b_G}$
6: $\mu_{0,0} \leftarrow 1, \mu_{0,1} \leftarrow 0, q_{0,1} \leftarrow q_{0,0}$
7: $a_0 \leftarrow \mathsf{Com}(\tilde{V}_y(q'_0, q_{0,0}); 0)$
8: SendToProver($(q'_0, q_{0,0})$) // see Line 5 of Figure 8
9: $d \leftarrow c$.depth
10:
11: **for** i=1,. . . ,d **do**
12: $(X, Y, r', r_0, r_1) \leftarrow$ ZK-SUMCHECKV($i, a_{i-1}, q'_{i-1}, q_{i-1,0}, q_{i-1,1}$)
13: // $X = \mathsf{Com}(v_0), Y = \mathsf{Com}(v_1)$
14:
15: **if** i<d **then**
16: // Pick the next random $\mu_{i,0}, \mu_{i,1}$ and
17: // compute random linear combination (§3.2)
18: $\mu_{i,0}, \mu_{i,1} \xleftarrow{R} \mathbb{F}$
19: $a_i \leftarrow X^{\mu_{i,0}} \odot Y^{\mu_{i,1}}$
20: $(q'_i, q_{i,0}, q_{i,1}) \leftarrow (r', r_0, r_1)$
21: SendToProver($\mu_{i,0}, \mu_{i,1}$) // see Line 14 of Figure 8
22:
23: // For the final check, reduce from two points to one point (§3.2)
24: $(\mathsf{Com}(H_0), \ldots, \mathsf{Com}(H_{b_G})) \leftarrow$ ReceiveFromProver() // see Line 18 of Figure 8
25: **for** $i = 0, \ldots, b_G$ **do**
26: EstablishKnowledgeOfOpening($\mathsf{Com}(H_i)$)
27: EstablishOpeningToSameValue($\mathsf{Com}(H_0), X$)
28: EstablishOpeningToSameValue($\mathsf{Com}(H_{b_G}) \odot \ldots \odot \mathsf{Com}(H_0), Y$)
29: $\tau \xleftarrow{R} \mathbb{F}$
30: SendToProver(τ) // see Line 23 of Figure 8
31: $q_d \leftarrow (r', (r_1 - r_0) \cdot \tau + r_0)$
32: $\zeta = \mathsf{Com}(H_{b_G})^{\tau^{\log G}} \odot \mathsf{Com}(H_{b_G-1})^{\tau^{\log G-1}} \odot \ldots \odot \mathsf{Com}(H_0)$
33: $T' \leftarrow \bigodot_{i=1}^{\sqrt{|w|}} T_i^{\check{\chi} i}$ // $\hat{\chi}$ is defined in Section 6
34: $R \leftarrow (\hat{\chi}_0, \hat{\chi}_{\sqrt{|w|}}, \ldots, \hat{\chi}_{\sqrt{|w|}\cdot(\sqrt{|w|}-1)})$ // $\hat{\chi}$ is defined in Section 6
35: EstablishDotProductRelationship($T'^{q_d[0]}, \zeta \oslash g^{(1-q_d[0])\tilde{V}_x(q_d[1,\ldots,b_N+b_G-1])}, R$)
36: **return** accept
37:
38: **function** ZK-SUMCHECKV(layer i,$a_{i-1}, q'_{i-1}, q_{i-1,0}, q_{i-1,1}$)
39: $(r', r_0, r_1) \xleftarrow{R} \mathbb{F}^{\log N} \times \mathbb{F}^{\log G} \times \mathbb{F}^{\log G}$
40: $r \leftarrow (r', r_0, r_1)$
41: **for** $j = 1, \ldots, \log N + 2 \log G$ **do**
42: $\alpha_j \leftarrow$ ReceiveFromProver() // α_j is $\mathsf{Com}(s_j)$; see Lines 19,47 of Figure 9
43: SendToProver($r[j]$) // see Lines 20,48 of Figure 9
44: $(X, Y, Z) \leftarrow$ ReceiveFromProver() // see Line 52 of Figure 9
45: // $X = \mathsf{Com}(v_0), Y = \mathsf{Com}(v_1), Z = \mathsf{Com}(v_0 v_1)$
46: // $\mathcal{V}$ computes $\{M_j\}$ as defined in Equation (5)
47: EstablishSumCheckRelation($a_{i-1}, \{\alpha_j\}, \{M_j\}, X, Y, Z$) // see Figure 1
48: **return** ($\mathsf{Com}(v_0), \mathsf{Com}(v_1), r', r_0, r_1$)

FIGURE 7—Pseudocode for $\mathcal{V}$ in Hyrax-Interactive (§7). $\mathcal{P}$'s work is described in Figures 8 and 9. For notational convenience, we assume $|x| = |w|$, as in Section 6.1.

1: **function** HYRAX-PROVE(ArithCircuit c, input x,, witness w)
2: // Commit to the rows of T via commitments $T_1, \ldots, T_{\sqrt{|w|}}$
3: SendToVerifier($T_1, \ldots, T_{\sqrt{|w|}}$) // see Line 3 of Figure 7
4: $b_N \leftarrow \log N, b_G \leftarrow \log G$
5: $(q'_0, q_{0,0}) \leftarrow$ ReceiveFromVerifier() // see Line 8 of Figure 7
6: $\mu_{0,0} \leftarrow 1, \mu_{0,1} \leftarrow 0, q_{0,1} \leftarrow q_{0,0}$
7: $a_0 \leftarrow \mathsf{Com}(\tilde{V}_y(q'_0, q_{0,0}); 0)$
8: $d \leftarrow c$.depth
9:
10: **for** i=1,. . . ,d **do**
11: $(X, Y, q'_i, q_{i,0}, q_{i,1}) \leftarrow$ ZK-SUMCHECKP($c, i, a_{i-1}, \alpha_{i-1,0}, \alpha_{i-1,1}$,
12: $q'_{i-1}, q_{i-1,0}, q_{i-1,1}$)
13: **if** i<d **then**
14: $(\mu_{i,0}, \mu_{i,1}) \leftarrow$ ReceiveFromVerifier() // see Line 21 of Figure 7
15: $a_i \leftarrow X^{\mu_{i,0}} \odot Y^{\mu_{i,1}}$
16:
17: // Compute Coefficients of the degree b_G polynomial H: $H_0, \ldots, H_{\log G}$
18: SendToVerifier($\mathsf{Com}(H_0), \ldots, \mathsf{Com}(H_{b_G})$) // see Line 21 of Figure 7
19: **for** $i = 0, \ldots, b_G$ **do**
20: EstablishKnowledgeOfOpening($\mathsf{Com}(H_i)$)
21: EstablishOpeningToSameValue($\mathsf{Com}(H_0), X$)
22: EstablishOpeningToSameValue($\mathsf{Com}(H_{b_G}) \odot \ldots \odot \mathsf{Com}(H_0), Y$)
23: $\tau \leftarrow$ ReceiveFromVerifier() // see Line 30 of Figure 7
24: $q_d \leftarrow (r', (r_1 - r_0) \cdot \tau + r_0)$
25: $\zeta = \mathsf{Com}(H_{b_G})^{\tau^{\log G}} \odot \mathsf{Com}(H_{b_G-1})^{\tau^{\log G-1}} \odot \ldots \odot \mathsf{Com}(H_0)$
26: $T' \leftarrow \bigodot_{i=1}^{\sqrt{|w|}} T_i^{\check{\chi} i}$ // $\hat{\chi}$ is defined in Section 6
27: $R \leftarrow (\hat{\chi}_0, \hat{\chi}_{\sqrt{|w|}}, \ldots, \hat{\chi}_{\sqrt{|w|}\cdot(\sqrt{|w|}-1)})$ // $\hat{\chi}$ is defined in Section 6
28: EstablishDotProductRelationship($T'^{q_d[0]}, \zeta \oslash g^{(1-q_d[0])\tilde{V}_x(q_d[1,\ldots,b_N+b_G-1])}, R$)

FIGURE 8—Pseudocode for $\mathcal{P}$ in Hyrax-Interactive (§7). The ZK-SumCheckP subroutine is defined in Figure 9. $\mathcal{V}$'s work is described in Figure 7. For notational convenience, we assume $|x| = |w|$, as in Section 6.1.

1: **function** ZK-SumcheckP(ArithCircuit c, layer i, a_{i-1},)
2: $\alpha_{i-1,0}, \alpha_{i-1,1}, q'_{i-1}, q_{i-1,0}, q_{i-1,1}$
3: **for** $j = 1, \ldots, b_N$ **do**
4: // In these rounds, prover sends commitment to degree-3 polynomial s_j
5: **for** all $\sigma \in \{0,1\}^{b_N - j}$ and $g \in \{0,1\}^{b_G}$ and $k \in \{-1, 0, 1, 2\}$ **do**
6: $s \leftarrow (g,\ g_L,\ g_R)$ // g_L, g_R are labels of g's layer-i inputs in sub-circuit.
7: termP $\leftarrow \widetilde{\text{eq}}(q'_{i-1}, r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j]) \cdot$
8: $\left(\alpha_{i-1,0} \cdot \chi_g(q_{i-1,0}) + \alpha_{i-1,1} \cdot \chi_g(q_{i-1,1})\right)$
9: termL $\leftarrow \tilde{V}_i\,(r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j],\ g_L)$
10: termR $\leftarrow \tilde{V}_i\,(r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j],\ g_R)$
11:
12: **if** g is an add gate **then** $s_j[\sigma, g][k] \leftarrow$ termP $\cdot$ (termL + termR)
13: **else if** g is a mult gate **then** $s_j[\sigma, g][k] \leftarrow$ termP $\cdot$ termL $\cdot$ termR
14:
15: **for** $k \in \{-1, 0, 1, 2\}$ **do**
16: $s_j[k] \leftarrow \sum_{\sigma \in \{0,1\}^{b_N - j}} \sum_{g \in \{0,1\}^{b_G}} s_j[\sigma, g][k]$
17:
18: // Compute coefficients of s_j and create a vector commitment (§5
19: SendToVerifier(Com(s_j)) // see Line 42 of Figure 7
20: $r'[j] \leftarrow$ ReceiveFromVerifier() // see Line 43 of Figure 7
21:
22: $r' \leftarrow (r'[1], \ldots, r'[b_N])$ // notation
23:
24: **for** $j = 1, \ldots, 2b_G$ **do**
25: // In these rounds, prover sends commitment to degree-2 polynomial $s_{b_N + j}$.
26: **for** all gates $g \in \{0,1\}^{b_G}$ and $k \in \{-1, 0, 1\}$ **do**
27: $s \leftarrow (g,\ g_L,\ g_R)$ // g_L, g_R are labels of g's layer-i inputs in subcircuit
28: $u_{k,0} \leftarrow (q_{i-1,0}[1], \ldots, q_{i-1,0}[b_G],\ r[1], \ldots, r[j-1],\ k)$
29: $u_{k,1} \leftarrow (q_{i-1,1}[1], \ldots, q_{i-1,1}[b_G],\ r[1], \ldots, r[j-1],\ k)$
30: termP $\leftarrow \widetilde{\text{eq}}(q'_{i-1}, r') \cdot \left(\alpha_{i,0} \cdot \prod_{\ell=1}^{b_G + j} \chi_{s[\ell]}(u_{k,0}[\ell]) +\right.$
31: $\left.\alpha_{i,1} \cdot \prod_{\ell=1}^{b_G + j} \chi_{s[\ell]}(u_{k,1}[\ell])\right)$
32:
33: **if** $j \le b_G$ **then**
34: termL $\leftarrow \tilde{V}_i\,(r', r[1], \ldots, r[j-1],\ k,\ g_L[j+1], \ldots, g_L[b_G])$
35: termR $\leftarrow \tilde{V}_i\,(r', g_R)$
36: **else** // $b_G < j \le 2b_G$
37: termL $\leftarrow \tilde{V}_i(r', r[1], \ldots, r[b_G])$
38: termR $\leftarrow \tilde{V}_i\,(r', r[b_G+1], \ldots, r[j-1],\ k,\ g_R[j-b_G+1], \ldots, g_R[b_G])$
39:
40: **if** g is an add gate **then** $s_{b_N+j}[g][k] \leftarrow$ termP $\cdot$ (termL + termR)
41: **else if** g is a mult gate **then** $s_{b_N+j}[g][k] \leftarrow$ termP $\cdot$ termL $\cdot$ termR
42:
43: **for** $k \in \{-1, 0, 1\}$ **do**
44: $s_{b_N+j}[k] \leftarrow \sum_{g \in \{0,1\}^{b_G}} s_{b_N+j}[g][k]$
45:
46: // Compute coefficients of s_{b_N+j} and create a vector commitment (§5)
47: SendToVerifier(Com(s_{b_N+j})) // see Line 42 of Figure 7
48: $r[j] \leftarrow$ ReceiveFromVerifier() // see Line 43 of Figure 7
49:
50: $r_0 \leftarrow (r[1], \ldots, r[b_G])$ $r_1 \leftarrow (r[b_G + 1], \ldots, r[2b_G])$ // notation
51: $v_0 \leftarrow \tilde{V}_i(r', r_0)$ $v_1 \leftarrow \tilde{V}_i(r', r_1)$ // X = Com(v_0), Y = Com(v_1), Z = Com($v_0 v_1$)
52: SendToVerifier(X, Y, Z) // see Line 44 of Figure 7
53: // $\mathcal{P}$ computes $\{M_k\}$ as defined in Equation (5).
54: EstablishSumCheckRelation(a_{i-1}, {Com(s_j)}, $\{M_k\}, X, Y, Z$) // see Figure 1
55:
56: **return** (Com(v_0), Com(v_1), r', r_0, r_1)

FIGURE 9—$\mathcal{P}$'s side of the zero-knowledge sum-check protocol in Hyrax-Interactive (§7).

1: **function** Verify(ArithCircuit c, input x, output y)
2: // $\mathcal{P}$ sends witness w to $\mathcal{V}$ in the clear at start of protocol.
3: $w \leftarrow$ ReceiveFromProver() // see Line 3 of Figure 12
4: $(q'_0,\ q_0) \xleftarrow{R} \mathbb{F}^{\log N} \times \mathbb{F}^{\log G}$
5: $a_0 \leftarrow \tilde{V}_y(q'_0, q_0)$ // $\tilde{V}_y$ is the multilin. ext. of the output y
6:
7: // The first iteration of sum-check only involves a claim about a single
8: // evaluation of $\tilde{V}_0$, rather than a linear combination of two evaluations;
9: // we encode this as a trivial linear combination.
10: $\mu_{0,0} \leftarrow 1, \mu_{0,1} \leftarrow 0, q_{0,1} \leftarrow q_0, q_{0,1} \leftarrow q_0$
11:
12: SendToProver(q'_0, q_0) // see Line 4 of Figure 12
13: $d \leftarrow$ c.depth
14:
15: **for** $i = 1, \ldots, d$ **do**
16: // Reduce $\mu_{i-1,0} \cdot \tilde{V}_{i-1}(q'_{i-1}, q_{i-1,0}) + \mu_{i-1,1} \cdot \tilde{V}_{i-1}(q'_{i-1}, q_{i-1,1}) \stackrel{?}{=} a_{i-1}$ to
17: // $Q_i(r', r_0, r_1) \stackrel{?}{=} e$, where Q_i is given in §3.2
18: $(e, r', r_0, r_1) \leftarrow$ SumCheckV($i,\ a_{i-1}$)
19: $(q'_i,\ q_{i,0}, q_{i,1}) \leftarrow (r',\ r_0, r_1)$
20:
21: **if** $i < d$ **then**
22: $v_0, v_1 \leftarrow$ ReceiveFromProver() // see Line 53 of Figure 13
23: termA1 $\leftarrow \mu_{i-1,0} \cdot \tilde{\text{add}}_i(q_{i-1,0}, r_0, r_1)$
24: termA2 $\leftarrow \mu_{i-1,1} \cdot \tilde{\text{add}}_i(q_{i-1,1}, r_0, r_1)$
25: termM1 $\leftarrow \mu_{i-1,0} \cdot \tilde{\text{mult}}_i(q_{i-1,0}, r_0, r_1)$
26: termM2 $\leftarrow \mu_{i-1,1} \cdot \tilde{\text{mult}}_i(q_{i-1,1}, r_0, r_1)$
27:
28: **if** $e \neq \widetilde{\text{eq}}(q'_{i-1},\ r') \cdot \left[\left(\text{termA1} + \text{termA2}\right) \cdot (v_0 + v_1)\right.$
29: $\left. + \left(\text{termM1} + \text{termM2}\right) \cdot v_0 \cdot v_1\right]$ **then**
30: **return** reject
31:
32: // Reduce the two v_0, v_1 questions to a random linear combination thereof
33: $\mu_{i,0}, \mu_{i,1} \xleftarrow{R} \mathbb{F}$
34: $a_i \leftarrow \mu_{i,0} \cdot v_0 + \mu_{i,1} \cdot v_1$
35:
36: SendToProver($\mu_{i,0}, \mu_{i,1}$) // see Line 12 of Figure 12
37:
38: // For the final layer, $\mathcal{P}$ and $\mathcal{V}$ reduce two points to one point (§3.2)
39: $(H_0, \ldots, H_{b_G}) \leftarrow$ ReceiveFromProver() // see Line 56 of Figure 13
40: $\tau \xleftarrow{R} \mathbb{F}$
41: $q_d \leftarrow (r', (r_1 - r_0) \cdot \tau + r_0)$
42: $a_d \leftarrow H_{b_G} \cdot \tau^{\log G} + H_{b_G - 1} \cdot \tau^{\log G - 1} + \ldots + H_0$
43:
44: // $\tilde{V}_d(\cdot)$ is the multilinear extension of (x, w) where x is input and w is witness
45: **if** $\tilde{V}_d(q'_d,\ q_d) \neq a_d$ **then**
46: **return** reject
47: **return** accept

FIGURE 10—$\mathcal{V}$'s side of our non-zero-knowledge interactive proof Gir^{++}. Gir^{++} is equivalent to the interactive proof in Giraffe, but incorporates a technique of Cheisa et al. [34] in place of GKR's "reducing from two points to one point" step. $\mathcal{V}$'s side of the sum-check protocol and $\mathcal{P}$'s work are described in Figures 11, 12, and 13.

```
1:  function SumCheckV(layer i, a_{i-1})
2:      e ← a_{i-1}
3:
4:      (r', r_0, r_1) ←R 𝔽^{b_N} × 𝔽^{b_G} × 𝔽^{b_G}
5:      r ← (r', r_0, r_1)
6:
7:      for j = 1, 2, ..., (b_N + 2b_G) do
8:
9:          // F_j is a degree-2 or degree-3 polynomial
10:         F_j ← ReceiveFromProver()    // see Lines 18,47 of Figure 13
11:
12:         if F_j(0) + F_j(1) ≠ e then
13:             return reject
14:
15:         SendToProver(r[j])    // see Lines 19,48 of Figure 13
16:
17:         e ← F_j(r[j])
18:
19:     return (e, r', r_0, r_1)
```

FIGURE 11—$\mathcal{V}$'s side of the sum-check protocol within Gir^{++}. This protocol reduces the claim that a_i equals the sum $\sum_{n,h_0,h_1} Q_i(n, h_0, h_1)$ (this sum equals $\alpha_{i-1,0} \cdot \tilde{V}_{i-1}(q'_{i-1}, q_{i-1,0}) + \alpha_{i-1,1} \cdot \tilde{V}_{i-1}(q'_{i-1}, q_{i-1,1})$, per Equation (3)) to the claim $e = Q_i(r', r_0, r_1)$.

```
1:  function Prove(ArithCircuit c, input x, output y)
2:      // Let w be a witness such that c(x, w) = y
3:      SendToVerifier(w)    // see Line 3 of Figure 10.
4:      (q'_0, q_0) ← ReceiveFromVerifier()    // see Line 12 of Figure 10
5:      μ_{0,0} ← 1, μ_{0,1} ← 0, q_{0,0} ← q_0, q_{0,1} ← q_0.
6:      d ← c.depth
7:
8:      // each circuit layer induces one sum-check invocation
9:      for i = 1, ..., d do
10:         (q'_i, q_{i,0}, q_{i,1}) ← SumCheckP(c, i, q'_{i-1}, μ_{i-1,0}, q_{i-1,0}, μ_{i-1,1}, q_{i-1,1})
11:         if i < d then
12:             (μ_{i,0}, μ_{i,1}) ← ReceiveFromVerifier()    // see Line 36 of Figure 10
```

FIGURE 12—Pseudocode for $\mathcal{P}$ in Gir^{++}. SumCheckP is defined in Figure 13.

1: **function** SumCheckP(ArithCircuit c, layer i, q'_{i-1}, $\mu_{i-1,0}$, $q_{i-1,0}$, $\mu_{i-1,1}$, $q_{i-1,1}$)
2: **for** $j = 1, \ldots, b_N$ **do**
3: // Prover sends degree-3 polynomial F_j. Does this by computing $F_j(-1), F_j(0), F_j(1), F_j(2)$ and then interpolating.
4:
5: **for** all $\sigma \in \{0,1\}^{b_N - j}$ and $g \in \{0,1\}^{b_G}$ and $k \in \{-1, 0, 1, 2\}$ **do**
6: $s \leftarrow (g,\ g_L,\ g_R)$ // g_L, g_R are labels of g's layer-i inputs in sub-circuit.
7:
8: termP $\leftarrow \widetilde{\text{eq}}(q'_{i-1}, r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j]) \cdot \left(\mu_{i-1,0} \cdot \chi_g(q_{i-1,0}) + \mu_{i-1,1} \cdot \chi_g(q_{i-1,1})\right)$
9: termL $\leftarrow \tilde{V}_i\,(r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j],\ g_L)$
10: termR $\leftarrow \tilde{V}_i\,(r'[1], \ldots, r'[j-1],\ k,\ \sigma[1], \ldots, \sigma[b_N - j],\ g_R)$
11:
12: **if** g is an add gate **then** $F_j[\sigma, g][k] \leftarrow$ termP $\cdot$ (termL + termR)
13: **else if** g is a mult gate **then** $F_j[\sigma, g][k] \leftarrow$ termP $\cdot$ termL $\cdot$ termR
14:
15: **for** $k \in \{-1, 0, 1, 2\}$ **do**
16: $F_j[k] \leftarrow \sum_{\sigma \in \{0,1\}^{b_N - j}} \sum_{g \in \{0,1\}^{b_G}} F_j[\sigma, g][k]$
17: // Use Lagrange interpolation to compute coefficients of F_j and send them to $\mathcal{V}$
18: SendToVerifier(F_j, 3) // see Line 10 of Figure 11
19: $r'[j] \leftarrow$ ReceiveFromVerifier() // see Line 15 of Figure 11
20:
21: $r' \leftarrow (r'[1], \ldots, r'[b_N])$ // notation
22:
23: **for** $j = 1, \ldots, 2b_G$ **do**
24: // In these rounds, prover sends degree-2 polynomial $F_{b_N + j}$.
25: **for** all gates $g \in \{0,1\}^{b_G}$ and $k \in \{-1, 0, 1\}$ **do**
26:
27: $s \leftarrow (g,\ g_L,\ g_R)$ // g_L, g_R are labels of g's layer-i inputs in subcircuit
28: $u_{k,0} \leftarrow (q_{i-1,0}[1], \ldots, q_{i-1,0}[b_G],\ r[1], \ldots, r[j-1],\ k)$
29: $u_{k,1} \leftarrow (q_{i-1,1}[1], \ldots, q_{i-1,1}[b_G],\ r[1], \ldots, r[j-1],\ k)$
30: termP $\leftarrow \widetilde{\text{eq}}(q'_{i-1}, r') \cdot \left(\mu_{i,0} \cdot \prod_{\ell=1}^{b_G + j} \chi_{s[\ell]}(u_{k,0}[\ell]) + \mu_{i,1} \cdot \prod_{\ell=1}^{b_G + j} \chi_{s[\ell]}(u_{k,1}[\ell])\right)$
31:
32: **if** $j \le b_G$ **then**
33: termL $\leftarrow \tilde{V}_i\,(r', r[1], \ldots, r[j-1],\ k,\ g_L[j+1], \ldots, g_L[b_G])$
34: termR $\leftarrow \tilde{V}_i\,(r', g_R)$
35: **else** // $b_G < j \le 2b_G$
36: termL $\leftarrow \tilde{V}_i(r', r[1], \ldots, r[b_G])$
37: termR $\leftarrow \tilde{V}_i\,(r', r[b_G+1], \ldots, r[j-1],\ k,\ g_R[j-b_G+1], \ldots, g_R[b_G])$
38:
39: **if** g is an add gate **then**
40: $F_{b_N + j}[g][k] \leftarrow$ termP $\cdot$ (termL + termR)
41: **else if** g is a mult gate **then**
42: $F_{b_N + j}[g][k] \leftarrow$ termP $\cdot$ termL $\cdot$ termR
43:
44: **for** $k \in \{-1, 0, 1\}$ **do**
45: $F_{b_N + j}[k] \leftarrow \sum_{g \in \{0,1\}^{b_G}} F_{b_N + j}[g][k]$
46: // Use Lagrange interpolation to compute coefficients of $F_{b_N + j}$ and send them to verifier
47: SendToVerifier($F_{b_N + j}$, 2) // see Line 10 of Figure 11
48: $r[j] \leftarrow$ ReceiveFromVerifier() // see Line 15 of Figure 11
49:
50: $r_0 \leftarrow (r[1], \ldots, r[b_G])$ $\quad$ $r_1 \leftarrow (r[b_G+1], \ldots, r[2b_G])$ // notation
51:
52: **if** $i < c$.depth **then**
53: SendToVerifier($\tilde{V}_i(r', r_0), \tilde{V}_i(r', r_1)$) // see Line 22 of Figure 10
54: **else** // in the last sum-check invocation, $\mathcal{P}$ and $\mathcal{V}$ reduce two points to one point (§3.2)
55: // First, compute coefficients of $H(\cdot)$, i.e., $\tilde{V}_d$ restricted to the line passing through (r', r_0) and (r', r_1).
56: SendToVerifier($(H_0, \ldots, H_{b_G})$) // see Line 39 of Figure 10
57:
58: **return** (r', r_0, r_1)

Figure 13—$\mathcal{P}$ pseudocode in our non-zero-knowledge interactive proof Gir^{++} for the layer-i sum-check invocation.