Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time

Jiaheng Zhang
UC Berkeley
jiaheng_zhang@berkeley.edu

Tianyi Liu
Texas A&M University
tianyi@tamu.edu

Weijie Wang
Shanghai Jiao Tong University
wangnick@sjtu.edu.cn

Yinuo Zhang
UC Berkeley
yinuo@berkeley.edu

Dawn Song
UC Berkeley
dawnsong@berkeley.edu

Xiang Xie
Shanghai Key Laboratory of
Privacy-Preserving Computation
xiexiang@matrixelements.com

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

ABSTRACT

We propose a new doubly efficient interactive proof protocol for general arithmetic circuits. The protocol generalizes the interactive proof for layered circuits proposed by Goldwasser, Kalai and Rothblum to arbitrary circuits, while preserving the optimal prover complexity that is strictly linear in the size of the circuits. The proof size remains succinct for low depth circuits and the verifier time is sublinear for structured circuits. We then construct a new zero knowledge argument scheme for general arithmetic circuits using our new interactive proof protocol together with polynomial commitments. Our key technique is a new sumcheck equation that reduces a claim about the output of one layer to claims about its input only, instead of claims about all the layers above which inevitably incurs an overhead proportional to the depth of the circuit. We developed efficient algorithms for the prover to run this sumcheck protocol and to combine multiple claims back into one in linear time in the size of the circuit.

Not only does our new protocol achieve optimal prover complexity asymptotically, but it is also efficient in practice. Our experiments show that it only takes 0.3 seconds to generate the proof for a circuit with more than 600,000 gates, which is 13 times faster than the original interactive proof protocol on the corresponding layered circuit. The proof size is 208 kilobytes and the verifier time is 66 milliseconds. Our implementation can take general arithmetic circuits directly, without transforming them to layered circuits with a high overhead on the size of the circuit.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11.
https://doi.org/10.1145/3460120.3484767

KEYWORDS

Interactive proofs; Zero knowledge proofs

ACM Reference Format:
Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. 2021. Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time . In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3460120.3484767

1 INTRODUCTION

Interactive proofs allow a powerful yet untrusted prover to convince a verifier through a sequence of interactions that the result of a computation is correctly computed. Since they were introduced by Goldwasser, Micali, and Rackoff [34] in the 1980s, interactive proofs have expanded people's understanding on traditional static mathematical proofs and led to many important theoretical results in complexity theory, such as IP=PSPACE [38, 44] and MIP=NEXP [11].

In recent years, there is great progress on turning interactive proofs from purely theoretical constructions to practical schemes with efficient implementations. In the seminal work of [33], Goldwasser, Kalai and Rothblum proposed doubly efficient interactive proofs where the prover can convince the verifier the correctness of the evaluation of a layered arithmetic circuit with addition gates and multiplication gates of fan-in two. The time for the prover to generate all the messages during the protocol (prover time) is a polynomial in the size of the circuit, and the time to validate the result (verifier time) is close to linear in the size of the input for log-space uniform circuits, thus the name "doubly efficient". The total communication between the prover and the verifier is only poly-logarithmic in the size of the circuit and linear in the depth of the circuit, which is *succinct* for bounded-depth circuits. We refer to the protocol in [33] as the *GKR* protocol . Later, researchers spent great effort improving the concrete efficiency of the GKR protocol. The prover time was improved to quasi-linear ($O(|C| \log |C|)$) in [24], and then to close to linear for various circuits with different structures [46, 48, 58]. Finally, in [53], Xie et al. proposed an algorithm to improve the prover time to strictly linear ($O(|C|)$) for

layered arithmetic circuits without assuming any structures, which is asymptotically optimal and very efficient in practice.

Another important advance of interactive proofs is using them to construct efficient zero knowledge argument schemes. In [56], Zhang et al. first proposed to combine the GKR protocol with polynomial commitments to build argument systems, where the prover can further prove to the verifier the computations on the prover's witness, without sending it directly to the verifier. Following the framework, there are many subsequent zero knowledge argument constructions based on interactive proofs, including [42, 50, 53, 55, 57]. These schemes demonstrate great prover efficiency and can achieve sublinear verifier time for structured circuits, thanks to the advantages of the interactive proofs and the GKR protocol.

Despite the progress of the GKR protocol, a major drawback is that the protocol only works for layered arithmetic circuits. Each gate can only connect to the layer above, due to the layer-by-layer reduction of the GKR protocol. In practice, it introduces a high overhead to pad general circuits to layered circuits using relay gates. Asymptotically, the circuit size increases from $O(|C|)$ to $O(d|C|)$ where $|C|$ is the size of the general circuit and d is the depth of the circuit. This is easily 1-2 orders of magnitude larger in practice as we show in our experiments, and introduces a big overhead on the prover time. Moreover, it is also inconvenient to implement circuits in a strictly layered way, and most existing tools such as rank-1-constraint-system (R1CS) cannot be used directly. Therefore, in this paper we ask the following question:

Is it possible to generalize the GKR protocol to directly support general circuits, without introducing any overhead on the prover?

1.1 Our Contributions

We answer the above question affirmatively by proposing a generalized doubly efficient interactive proofs for arbitrary arithmetic circuits, where each gate can take the output of any gate as input. The prover time is still linear in the size of the circuit, and is very efficient in practice. In particular, our contributions are:

- We generalize the GKR protocol to work on arbitrary arithmetic circuits efficiently for the first time. For a general circuit of size $|C|$ and depth d, the prover time is $O(|C|)$, the same as the original GKR protocol on a layered circuit with the same size. The overhead on the proof size and the verifier time is minimal. The proof size in our new protocol is $\min\{O(d \log |C| + d^2), O(|C|)\}$. For structured circuits, the verifier time is also $\min\{O(d \log |C| + d^2), O(|C|)\}$. Those in the original GKR are $\min\{O(d \log |C|), O(|C|)\}$.
- Together with zero knowledge polynomial commitments, we construct zero knowledge arguments for general arithmetic circuits. The zero knowledge version of our interactive proof protocols does not incur any overhead asymptotically on the prover time, the proof size and the verifier time compared to the plain version.
- We fully implement a system, Virgo++, for our new interactive proof protocols and zero knowledge arguments. We show that on random circuits with $d = 50$ and $d = 75$, our new protocols are 9-13× faster than the state-of-the-art GKR protocol on the corresponding layered circuits. The prover time per gate (the constant in the linear complexity) is only 1.3× more than the original GKR protocol on layered circuits. Therefore, as long as padding the general circuit to layered circuit makes the size 1.3× or larger, our new protocol will have faster prover time. The verifier time of our new protocols is 17-25× faster, while the proof size is only slightly larger than GKR on layered circuits.

1.2 Technical Overview

The key idea of the GKR protocol is to write the values in the i-th layer of the circuit as an equation of the values in the previous layer $i + 1$:

$$\tilde{V}_i(z) = \sum_{x,y} (\tilde{add}_{i+1}(z,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + \tilde{mult}_{i+1}(z,x,y)\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)),$$

where $\tilde{V}_i()$ and $\tilde{V}_{i+1}()$ are polynomials defined by the values in layer i and layer $i + 1$ of the circuit, and $\tilde{add}_{i+1}()$ and $\tilde{mult}_{i+1}()$ are polynomials describing the addition/multiplication gates and their connections in these two layers. See Section 2.2 for the formal definitions and equations. With this equation, the GKR protocol invokes the sumcheck protocol (See Section 2.2) to reduce the correctness of $\tilde{V}_i()$ to the correctness of $\tilde{V}_{i+1}()$, which can be further reduced to $\tilde{V}_{i+2}()$ using the same equation and protocol for layer $i + 1$. Therefore, starting from the output layer, $\mathcal{P}$ and $\mathcal{V}$ perform the reduction layer by layer to the input layer, which can be validated by $\mathcal{V}$ directly. The prover time is linear in the number of gates in each layer [53], and thus the total prover time is linear in the size of the circuit $|C|$.

The above equation relies on the fact that gates in layer i can only take input from gates in layer $i + 1$. To generalize the GKR protocol to arbitrary circuits, as each gate in layer i can take input from any gate in layer j for $j > i$, $\tilde{V}_i()$ becomes:

$$\tilde{V}_i(z) = \sum_{x,y} \sum_{j,k>i} (\tilde{add}_{i,j,k}(z,x,y)(\tilde{V}_j(x) + \tilde{V}_k(y)) + \tilde{mult}_{i,j,k}(z,x,y)\tilde{V}_j(x)\tilde{V}_k(y)).$$

Namely, we have multiple summations, one for each layer above. $\mathcal{P}$ and $\mathcal{V}$ can still run the sumcheck protocol on the new equation as before to reduce the correctness of the output to the input. However, the prover time becomes $O(d|C|)$, as there are d polynomials in the sum and the total size is $O(d|C|)$. This is as bad as padding the general circuit to a layered circuit.

We introduce two new techniques to reduce the prover time to linear. First, we observe that the key reason why the prover time of the sumcheck protocol on the generalized equation above becomes $O(d|C|)$ is that the polynomials $\tilde{V}_j()$, $\tilde{V}_k()$ of the entire layers j, k for all $j, k > i$ are used. Merely writing out all the polynomials takes $O(d|C|)$ time. Instead, we propose a new way to write $\tilde{V}_i()$ as a equation of the polynomials defined by *only those values used by layer i* from layer $j > i$. As each gate has only two inputs, the total size of the all these polynomials is bounded by two times the number of gates in layer i. We then design a new algorithm for the prover to run the sumcheck protocol on the new equation with linear time by utilizing the sparsity of the polynomials. Therefore, the prover time of this step is reduced to $O(|C|)$. The formal algorithms are presented in Section 3.2.

Second, we face a new challenge when executing the sumcheck protocol on the equation above for general circuits. At the end of the protocol, the correctness of $\tilde{V}_i()$ is reduced to multiple polynomials

for all layers above. Therefore, when proceeding to the next layer $i+1$, the verifier has received $i+1$ claims about $\tilde{V}_{i+1}()$, one from the sumcheck protocol of each layer below. Naively combining these claims into one would incur a prover time of $O(d|C|)$. In this paper, we propose a new technique to combine these claims efficiently. The key observation is that as all the claims are about polynomials defined by subsets of values in layer $i+1$, the claims can be computed using a layered arithmetic circuit. The inputs of the circuit are the values in layer $i+1$, and the outputs are these claims. By running the original GKR protocol on the layered circuit, we are able to reduce these claims to a single claim of $\tilde{V}_{i+1}()$. With proper design, we are able to bound the total size of this circuit in all rounds by $O(|C|)$. Therefore, the prover complexity in this step is also $O(|C|)$. See Figure 2 and Section 3.3 for more details.

Furthermore, inspired by the structure of this circuit, we are able to design a single sumcheck protocol to combine multiple claims of $\tilde{V}_{i+1}$ into one. This second approach further improves the prover time, the proof size and the verifier time. Putting the two steps together, we are able to construct a generalized GKR protocol for arbitrary arithmetic circuits while maintaining the optimal prover time of $O(|C|)$.

1.3 Related Work

Interactive proofs were formalized by Goldwasser, Micali, and Rackoff in [34]. In the seminal work of [33], Goldwasser et al. proposed the doubly efficient interactive proof for layered arithmetic circuits. Later, Cormode et al. improved the prover time of the GKR protocol from $O(|C|^3)$ to $O(|C|\log|C|)$ using multilinear extensions instead of low degree extensions in [24]. Several follow-up papers further reduce the prover time for circuits with special structures [46, 47, 58]. Eventually, Xie et al. [53] proposed a variant of the GKR protocol with $O(|C|)$ prover time for arbitrary layered arithmetic circuits. All these existing works follow the layered structure of the GKR protocol and doubly efficient interactive proofs for general arithmetic circuits have not been considered before.

In [56], Zhang et al. extended the GKR protocol to an argument system using polynomial commitments. Subsequent works [42, 50, 53, 55, 57] followed the framework and constructed efficient zero knowledge argument schemes based on interactive proofs. We follow the approach of [23, 53, 55] and constructs zero knowledge arguments for general circuits instead of layered circuits. Notably, there is a recent work [42] on constructing interactive proof-based zero knowledge arguments for R1CS. The protocol reduces the R1CS to a polynomial commitment on the entire extended witness of all the values in the circuit using one sumcheck. On the contrary, the GKR protocols reduce the evaluation of the circuit to only the input of the circuit. As the polynomial commitments are usually the overhead of the zero knowledge argument schemes, we expect that our scheme has faster prover time, while the scheme in [42] has smaller proof size. We give detailed comparisons in Section 5.2. In addition, the scheme in [42] cannot be used for delegation of computations without cryptographic assumptions, which is the original goal of the GKR protocols. In a recent manuscript [43], the proof size of the scheme in [42] is improved from square-root to logarithmic in the size of the R1CS instance, but the prover time is 3.8× slower. In a different setting, Blumberg et al. [18] construct argument schemes using interactive proofs with two provers.

There is a rich literature of zero knowledge arguments other than schemes based on interactive proofs. Categorized by their underlying techniques, there are schemes based on quadratic arithmetic programs (QAP) [14, 15, 26, 29, 40, 49, 51], interactive oracle proofs (IOP) [14, 16], discret-log [13, 19, 21, 35], MPC-in-the-head [10, 22, 31, 36] and lattice [12]. We refer the readers to surveys [51] and recent papers [55] on zero knowledge proofs and arguments for a more comprehensive list of schemes.

2 PRELIMINARIES

We use $\mathsf{negl}(\cdot): \mathbb{N} \to \mathbb{N}$ to denote the negligible function, where for each positive polynomial $f(\cdot)$, $\mathsf{negl}(k) < \frac{1}{f(k)}$ for sufficiently large integer k. Let λ denote the security parameter. "PPT" stands for probabilistic polynomial time. We use $f(), g()$ for polynomials, x, y, z for vectors of variables and g, u, v for vectors of values. x_i denotes the i-th variable in x. We use bold letters such as $\mathbf{A}$ to represent arrays. For a multivariate polynomial f, its "variable-degree" is the maximum degree of f in any of its variables.

2.1 Interactive Proofs

Interactive proofs. An interactive proof allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of some statement. The interactive proof runs in several rounds, allowing $\mathcal{V}$ to ask questions in each round based on $\mathcal{P}$'s answers of previous rounds. We phrase this in terms of $\mathcal{P}$ trying to convince $\mathcal{V}$ that $C(x) = y$. We formalize interactive proofs in the following:

Definition 2.1. Let C be a function. A pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an interactive proof for C with soundness ϵ if the following holds:

- **Completeness.** For every x such that $C(x) = y$ it holds that $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = accept] = 1$.
- **ϵ-Soundness.** For any x with $C(x) \neq y$ and any $\mathcal{P}^*$ it holds that $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle = accept] \leq \epsilon$

We say an interactive proof scheme has succinct proof size and succinct verifier time if both of them are $O(\text{polylog}(|C|, |x|))$.

2.2 Doubly Efficient Interactive Proofs for Layered Circuits

In [33], Goldwasser et al. proposed an efficient interactive proof protocol for layered arithmetic circuits. We present the details of the protocol here.

2.2.1 Sumcheck Protocol. The GKR protocol uses the sumcheck protocol as a major building block. The problem is to sum a multivariate polynomial $f: \mathbb{F}^\ell \to \mathbb{F}$ on the Boolean hypercube: $H = \sum_{b_1, b_2, \ldots, b_\ell \in \{0,1\}} f(b_1, b_2, \ldots, b_\ell)$. Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \ldots, b_\ell$. Lund et al. [38] proposed a *sumcheck* protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ of the correct sum, denoted by H. We provide the sumcheck protocol in Protocol 1. The proof size of the protocol is $O(\tau\ell)$, where τ is the variable-degree of f, as in each round, $\mathcal{P}$ sends a univariate polynomial of

Protocol 1 (Sumcheck). *The protocol proceeds in ℓ rounds.*

- *In the first round, $\mathcal{P}$ sends a univariate polynomial*

$$f_1(x_1) \stackrel{def}{=} \sum_{b_2,\ldots,b_\ell \in \{0,1\}} f(x_1, b_2, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $H = f_1(0) + f_1(1)$. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.
- *In the i-th round, where $2 \le i \le \ell - 1$, $\mathcal{P}$ sends a univariate polynomial*

$$f_i(x_i) \stackrel{def}{=} \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$, and sends a random challenge $r_i \in \mathbb{F}$ to $\mathcal{P}$.
- *In the ℓ-th round, $\mathcal{P}$ sends a univariate polynomial*

$$f_\ell(x_\ell) \stackrel{def}{=} f(r_1, r_2, \ldots, r_{l-1}, x_\ell),$$

  $\mathcal{V}$ checks $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. The verifier generates a random challenge $r_\ell \in \mathbb{F}$. Given oracle access to an evaluation $f(r_1, r_2, \ldots, r_\ell)$ of f, $\mathcal{V}$ will accept if and only if $f_\ell(r_\ell) = f(r_1, r_2, \ldots, r_\ell)$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

Figure 1: The sumcheck protocol.

one variable in f, which can be uniquely defined by $\tau + 1$ points. The verifier time of the protocol is $O(\tau\ell)$. The prover time depends on the degree and the sparsity of f, and we will give the complexity later in our scheme. The sumcheck protocol is complete and sound with $\epsilon = \frac{\tau\ell}{|\mathbb{F}|}$.

*Definition 2.2 (**Multi-linear Extension**).* Let $V : \{0,1\}^\ell \to \mathbb{F}$ be a function. The *multilinear extension* of V is the unique polynomial $\tilde{V} : \mathbb{F}^\ell \to \mathbb{F}$ such that $\tilde{V}(x_1, x_2, ..., x_\ell) = V(x_1, x_2, ..., x_\ell)$ for all $x_1, x_2, \ldots, x_\ell \in \{0,1\}$. $\tilde{V}$ can be expressed as:

$$\tilde{V}(x_1, x_2, ..., x_\ell) = \sum_{b \in \{0,1\}^\ell} \prod_{i=1}^{\ell} ((1-x_i)(1-b_i) + x_i b_i)) \cdot V(b),$$

where b_i is i-th bit of b.

Multilinear extensions of arrays. Inspired by the closed-form equation of the multilinear extension given above, we can view an array $\mathbf{A} = (a_0, a_1, \ldots, a_{n-1})$ as a function $A : \{0,1\}^{\log n} \to \mathbb{F}$ such that $\forall i \in [0, n-1], A(i_1, \ldots, i_{\log n}) = a_i$. Here we assume n is a power of 2. Therefore, in this paper, we abuse the use of multilinear extension on an array as the multilinear extension $\tilde{A}$ of A.

*Definition 2.3 (**Identity function**).* Let $\beta : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be the identity function such that $\beta(x, y) = 1$ if $x = y$, and $\beta(x, y) = 0$ otherwise. Suppose $\tilde{\beta}$ is the multilinear extension of β. Then $\tilde{\beta}$ can be expressed as: $\tilde{\beta}(x, y) = \prod_{i=1}^{\ell}((1 - x_i)(1 - y_i) + x_i y_i)$.

GKR Protocol. Using the sumcheck protocol as a building block, Goldwasser et al. [33] showed an interactive proof protocol for layered arithmetic circuits. Let C be a layered arithmetic circuit with depth d over a finite field $\mathbb{F}$. Each gate in the i-th layer takes inputs from two gates in the $(i+1)$-th layer; layer 0 is the output layer and layer d is the input layer. The protocol proceeds layer by layer. Upon receiving the claimed output from $\mathcal{P}$, in the first round, $\mathcal{V}$ and $\mathcal{P}$ run the sumcheck protocol to reduce the claim about the output to a claim about the values in the layer above. In the i-th round, both parties reduce a claim about layer $i-1$ to a claim about layer i through the sumcheck protocol. Finally, the protocol terminates with a claim about the input layer d, which can be checked directly by $\mathcal{V}$. If the check passes, $\mathcal{V}$ accepts the claimed output.

Notation. We follow the convention in prior works of GKR protocols [24, 46, 53, 55, 56]. We denote the number of gates in the i-th layer as S_i and let $s_i = \lceil \log S_i \rceil$. (For simplicity, we assume S_i is a power of 2, and we can pad the layer with dummy gates otherwise.) We then define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes a binary string $b \in \{0,1\}^{s_i}$ and returns the output of gate b in layer i, where b is called the gate label. With this definition, V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. Finally, we define two additional functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, referred to as *wiring predicates* in the literature. add_i ($mult_i$) takes one gate label $z \in \{0,1\}^{s_{i-1}}$ in layer $i-1$ and two gate labels $x, y \in \{0,1\}^{s_i}$ in layer i, and outputs 1 if and only if gate z is an addition (multiplication) gate that takes the output of gate x, y as input. With these function definitions and their multilinear extensions, $\tilde{V}_i$ can be written as:

$$\tilde{V}_i(g) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} \tilde{add}_{i+1}(g, x, y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + \tilde{mult}_{i+1}(g, x, y)\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y), \quad (1)$$

where $g \in \mathbb{F}^{s_i}$ is a random vector.

Protocol. With Equation 1, the GKR protocol proceeds as following. The prover $\mathcal{P}$ first sends the claimed output of the circuit to $\mathcal{V}$. From the claimed output, $\mathcal{V}$ defines polynomial $\tilde{V}_0$ and computes $\tilde{V}_0(g)$ for a random $g \in \mathbb{F}^{s_0}$. $\mathcal{V}$ and $\mathcal{P}$ then invoke a sumcheck protocol on Equation 1 with $i = 0$. As described in Section 2.2.1, at the end of the sumcheck, $\mathcal{V}$ needs an oracle access to $f_i(g, u, v)$, where u, v are randomly selected in $\mathbb{F}^{s_{i+1}}$. To compute $f_i(g, u, v)$, $\mathcal{V}$ computes $\tilde{add}_{i+1}(g, u, v)$ and $\tilde{mult}_{i+1}(g, u, v)$ locally (they only depend on the wiring pattern of the circuit, not on the values), asks $\mathcal{P}$ to send $\tilde{V}_1(u)$ and $\tilde{V}_1(v)$ and computes $f_i(g, u, v)$ to complete the sumcheck protocol. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduce a claim about the output to two claims about values in layer 1. $\mathcal{V}$ and $\mathcal{P}$ could invoke two sumcheck protocols on $\tilde{V}_1(u)$ and $\tilde{V}_1(v)$ recursively to layers above, but the number of invocations would increase exponentially.

Combining two claims using a random linear combination. One way to combine two claims $\tilde{V}_i(u)$ and $\tilde{V}_i(v)$ is using random linear combinations, as proposed in [23, 50]. Upon receiving the two claims $\tilde{V}_i(u)$ and $\tilde{V}_i(v)$, $\mathcal{V}$ selects $\alpha_{i,1}, \alpha_{i,2} \in \mathbb{F}$ randomly and computes $\alpha_{i,1}\tilde{V}_i(u) + \alpha_{i,2}\tilde{V}_i(v)$. Based on Equation 1, this random linear combination can be written as

$$\alpha_{i,1}\tilde{V}_i(u) + \alpha_{i,2}\tilde{V}_i(v) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} \Big((\alpha_{i,1}\tilde{add}_{i+1}(u, x, y) + \alpha_{i,2}\tilde{add}_{i+1}(v, x, y)) (\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + (\alpha_{i,1}\tilde{mult}_{i+1}(u, x, y) + \alpha_{i,2}\tilde{mult}_{i+1}(v, x, y)) \tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y) \Big). \quad (2)$$

$\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 2 instead of Equation 1. At the end of the sumcheck protocol, $\mathcal{V}$ still receives two claims about $\tilde{V}_{i+1}$, computes their random linear combination and proceeds to the layer above recursively until the input layer.

The formal GKR protocol is presented in Protocol 2 in Appendix A. With the optimal algorithms with a linear prover time proposed in [53], we have the following theorem:

Theorem 2.4. *[53]. Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a d-depth layered arithmetic circuit. Protocol 2 is an interactive proof for the function computed by C with soundness $O(d \log |C|/|\mathbb{F}|)$. It uses $O(d \log |C|)$ rounds of interaction and the running time of the prover $\mathcal{P}$ is $O(|C|)$. Let T be the time to evaluate all $\tilde{add}_i$ and $\tilde{mult}_i$ at the corresponding random points, the running time of $\mathcal{V}$ is $O(n + k + d \log |C| + T)$.*

3 GENERALIZING GKR TO ARBITRARY ARITHMETIC CIRCUITS

Though the GKR protocol has great prover efficiency as demonstrated in [46, 48, 53, 55] and is used as a major building block to construct fast zero knowledge proof schemes, one major drawback is that the protocol only works for layered arithmetic circuits, i.e., each gate can only take input from the layer above. In this section, we show how to generalize the GKR protocol to arbitrary circuits with no overhead on the prover time.

We consider a general arithmetic circuit C with fan-in 2, which can be viewed as a directed acyclic graph (DAG), G_C. Each gate in C is a vertex in G_C and each wire is a directed edge in G_C. The in-degree of each vertex is at most 2. The depth of the circuit d is defined as the length of the longest path in the DAG. Without loss of generality, we assume that all input gates are at layer d, and all output gates are at layer 0.[1] Following the order to evaluate the circuit, we can actually assign a layer number to each gate topologically. In particular, $\text{layer}(g) = d$ if g is an input gate of the circuit; if gate g is not an input, suppose gate u and gate v are the input gates of g, then $\text{layer}(g) = \min(\text{layer}(u), \text{layer}(v)) - 1$, where the function $\text{layer}(x)$ represents the layer of the gate x and min is the minimum. Because of this definition, an interesting observation is that a gate at layer i must take at least one input from layer $i+1$, otherwise it cannot be labeled as in layer i. Also obviously, a gate at layer i can only take input from layer j such that $j > i$.

Same as the original GKR protocol, we use S_i as the number of gates in the i-th layer and $s_i = \lceil \log S_i \rceil$. For simplicity, we assume S_i is a power of 2, and we can pad the layer with dummy gates otherwise. The function V_i takes a binary string b and outputs the b-th gate value in layer i of C. As now every gate can take input from any layer above, we generalize the notations naturally and define $add_{i,j}, mult_{i,j} : \{0,1\}^{s_{i-1}, s_i, s_j} \to \{0,1\}$ as the wiring-predicate functions for the general circuit C. $add_{i,j}$ takes one gate label $z \in \{0,1\}^{s_{i-1}}$ in layer $i-1$, one gate label $x \in \{0,1\}^{s_i}$ in layer i and one gate label $y \in \{0,1\}^{s_j}$ in layer j for $j \geq i$, and outputs 1 if and only if gate z is an addition gate that takes the output of gate x, y as input. $mult_{i,j}$ is defined similarly for multiplication gates. $\tilde{f}$ represents the multilinear extensions of the function f.

3.1 A Naive Generalization of GKR

With these definitions, we first describe a naive generalization of the GKR protocol to general arithmetic circuits. We follow the core idea of the GKR protocol to reduce the claim about V_i layer by layer via the sumcheck protocol. In a general circuit, a gate in layer i can take the output of any gate in layer $i+1$ to d, thus we simply extend Equation 1 to have one *add* and one *mult* for each layer above. Recall from above that every gate at layer i must have at least one input from layer $i+1$, we assume that this is the left input and rewrite the sumcheck equation in Equation 1 as:

$$\tilde{V}_i(g) = \sum_{j=i+1}^{d} \sum_{x \in \{0,1\}^{s_{i+1}}, y \in \{0,1\}^{s_j}} \Big(\tilde{add}_{i+1,j}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_j(y)) + \tilde{mult}_{i+1,j}(g,x,y)\tilde{V}_{i+1}(x)\tilde{V}_j(y) \Big), \quad (3)$$

for any $g \in \mathbb{F}^{s_i}$. With this equation, starting from the output layer, in round i, the first step is that $\mathcal{P}$ and $\mathcal{V}$ engage the sumcheck protocol on Equation 3 to reduce one claim about layer i to claims about previous layers. At the end of the sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ evaluations of $\tilde{V}_{i+1}(u), \tilde{V}_{i+1}(v), \tilde{V}_{i+2}(v), \ldots, \tilde{V}_d(v)$ on the randomness of u and v. $\mathcal{V}$ evaluates all $\tilde{add}$ and $\tilde{mult}$ on her own and completes the last round of the sumcheck protocol.

In the second step, when going to a new layer, $\mathcal{P}$ and $\mathcal{V}$ need to combine multiple claims about this layer. Here in the naive approach, we use the same method of random linear combinations. When reaching layer i, $\mathcal{V}$ has received the claims about $\tilde{V}_i$ from layer $0, 1, \ldots, i-1$ (twice from layer $i-1$). Denote the randomness of these claims as $g^{(0)}, g^{(1)}, \ldots, g^{(i)}$. $\mathcal{V}$ picks a random number $\alpha_{i,k}$ for each claim, and we can rewrite Equation 3 as:

$$\sum_{k=0}^{i} \alpha_{i,k} \tilde{V}_i(g^{(k)}) = \sum_{k=0}^{i} \alpha_{i,k} \sum_{j=i+1}^{d} \sum_{x \in \{0,1\}^{s_{i+1}}, y \in \{0,1\}^{s_j}} \Big(\tilde{add}_{i+1,j}(g^{(k)},x,y) (\tilde{V}_{i+1}(x) + \tilde{V}_j(y)) + \tilde{mult}_{i+1,j}(g^{(k)},x,y)\tilde{V}_{i+1}(x)\tilde{V}_j(y) \Big). \quad (4)$$

$\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 4 instead of Equation 3. At the end of the sumcheck protocol, $\mathcal{V}$ still receives claims about $\tilde{V}_{i+1}, \tilde{V}_{i+2}, \ldots, \tilde{V}_d$. For layer $i+1$, $\mathcal{V}$ computes their random linear combination and proceeds to the sumcheck protocol for layer $i+1$ recursively.

This protocol is a direct generalization of the GKR protocol in Protocol 2, and it is not hard to see that the protocol is sound. Unfortunately, it introduces a big overhead on the prover time. First, in the beginning of the sumcheck protocol on Equation 3, the equation is defined over the multilinear extensions $\tilde{V}_{i+1}, \tilde{V}_{i+2}, \ldots, \tilde{V}_d$. Hence, the prover time in this step is at least $O(S_{i+1} + S_{i+2} + \ldots + S_d)$. In fact, merely listing these polynomials and evaluating them at random points already take $O(S_{i+1} + S_{i+2} + \ldots + S_d)$ time, not to mention the prover time of the sumcheck protocol. Therefore, the total prover time is $O(dS_d + (d-1)S_{d-1} + \ldots + S_1) = O(d|C|)$ for all layers. There is a multiplicative overhead of d on the prover time, which is in fact as bad as transforming the general circuit to a layered circuit. Second, in the step of random linear combinations, as shown in Equation 4, $\mathcal{V}$ combines $i+1$ claims together for layer i. On the right hand side of the equation, each $\tilde{add}$ and $\tilde{mult}$ has to be evaluated on $i+1$ different random points $g^{(k)}$. This again introduces a prover time of $O(d|C|)$. Therefore, overall the prover time of this naive generalized GKR protocol is $O(d|C|)$, as slow as naively transforming the general circuit to a layered circuit.

In the next two subsections, we will show how to remove the overhead of each of the two steps.

[1] Note that as we support general circuits, it takes at most one relay gate per input/output to transform an arbitrary circuit to a circuit with such property. Thus the overhead is small and we assume so for simplicity.

3.2 Sumcheck with Linear Prover Time

As explained above, the main overhead of the sumcheck on Equation 3 in the first step comes from the fact that each layer can connect to all layers above in a general circuit, and defining $\tilde{V}_{i+1}$, $\tilde{V}_{i+2}, \ldots, \tilde{V}_d$ already blows up the complexity. Therefore, instead of using the multilinear extension of the entire layer, we define the multilinear extension of *only those gates used in layer i* from a previous layer. As each gate only has two input gates, there are at most $2S_i$ gates connecting to gates in layer i in total. In this way, the total size of these multilinear extensions is bounded by $O(S_i)$.

Formally speaking, we also generalize the definitions of S and s such that $S_{i,j}$ denotes the number of gates connecting from layer j ($j > i$) to layer i, and $s_{i,j} = \lceil \log S_{i,j} \rceil$. We then introduce a new function $V_{i,j} : \{0,1\}^{s_{i,j}} \to \mathbb{F}$, which is defined by the subset of gates from layer j connecting to layer i in a pre-defined order. The function takes a binary string $b \in \{0,1\}^{s_{i,j}}$ and returns the b-th value in this subset. We also re-define $add_{i,j}, mult_{i,j} : \{0,1\}^{s_{i-1}+s_{i-1,i}+s_{i-1,j}} \to \{0,1\}$ to take labels from the subsets instead of the labels of the entire layers. In particular, $add_{i,j}(z,x,y) = 1$ ($mult_{i,j}(z,x,y) = 1$) if and only if gate z in layer $i-1$ is the addition (multiplication) of value $V_{i-1,i}(x)$ and $V_{i-1,j}(y)$. With multilinear extensions of these functions, we rewrite Equation 3 as

$$\tilde{V}_i(g) = \sum_{j=i+1}^{d} \sum_{x \in \{0,1\}^{s_{i,i+1}}, y \in \{0,1\}^{s_{i,j}}} \Big(\tilde{add}_{i+1,j}(g,x,y)(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,j}(y)) + \tilde{mult}_{i+1,j}(g,x,y)\tilde{V}_{i,i+1}(x)\tilde{V}_{i,j}(y) \Big). \quad (5)$$

Unlike Equation 3, in Equation 5, the size of $\tilde{V}_{i,i+1}, \ldots, \tilde{V}_{i,d}$ are bounded by $O(S_i)$. Moreover, the $\tilde{add}$ and $\tilde{mult}$ polynomials are still sparse. In fact, the total number of nonzeros in all $\tilde{add}$ and $\tilde{mult}$ together is S_i. Therefore, using similar ideas proposed in [53], we are able to develop an algorithm for the prover to run the sumcheck in linear time $O(S_i)$, instead of $O(S_i + S_{i+1} + \ldots + S_d)$.

Before presenting the linear-time algorithm, we make one more refinement on the equation. Note that Equation 5 consists of multiple sums, because the number of gates connecting from layer $j > i$ to layer i is different for each j. We cannot pad them to the same length, as it would introduce an overhead asymptotically. We combine them into a single sum in the following way. Without loss of generality, let $s_{i,i+1}$ be the largest. With the help of the identity function, we rewrite Equation 5 as:

$$\tilde{V}_i(g) = \sum_{x,y \in \{0,1\}^{s_{i,i+1}}} \sum_{j=i+1}^{d} \Big(\tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) \big(\tilde{add}_{i+1,j}(g,x,y_1,\ldots,y_{s_{i,j}})(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,j}(y_1,\ldots,y_{s_{i,j}})) + \tilde{mult}_{i+1,j}(g,x,y_1,\ldots,y_{s_{i,j}})\tilde{V}_{i,i+1}(x)\tilde{V}_{i,j}(y_1,\ldots,y_{s_{i,j}}) \big) \Big) \quad (6)$$

Note that the only difference between Equation 5 and 6 is that in Equation 6 all the sums are over $y \in \{0,1\}^{s_{i,i+1}}$, the longest binary string. For $j = i+2, \ldots, d$, as $\tilde{add}_{i+1,j}$, $\tilde{mult}_{i+1,j}$ and $\tilde{V}_{i,j}$ only take $y_1, \ldots, y_{s_{i,j}}$, we multiply each term with $\tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1})$. This guarantees that the term only appears once in the sum when $y_{s_{i,j}+1} = \ldots = y_{s_{i,i+1}} = 1$, as $\tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) = y_{s_{i,j}+1} \cdot \ldots \cdot y_{s_{i,i+1}}$. Thus Equation 6 holds. In this way, we do not have to pad all the polynomials to the same size. We only pad the size of each subset to the nearest power of 2, which incurs at most an overhead of 2.

Algorithm 1 $\{a_1, \ldots, a_\ell\} \leftarrow$ SumCheckProduct $(f, \mathbf{A}_f, g, \mathbf{A}_g, r_1, \ldots, r_\ell)$

Input: Multilinear polynomials f and g, bookkeeping tables $\mathbf{A}_f$ and $\mathbf{A}_g$, random challenges $r_1, \ldots, r_\ell$;
Output: ℓ sumcheck messages for $\sum_{x \in \{0,1\}^\ell} f(x)g(x)$. Each message a_i consists of 3 elements (a_{i0}, a_{i1}, a_{i2});

for $i = 1, \ldots, \ell$ **do**
  for $b \in \{0,1\}^{\ell-i}$ **do** *// b is both a number and its binary representation.*
    for $t = 0, 1, 2$ **do**
      Let $f(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_f[b] \cdot (1-t) + \mathbf{A}_f[b + 2^{\ell-i}] \cdot t$
      Let $g(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_g[b] \cdot (1-t) + \mathbf{A}_g[b + 2^{\ell-i}] \cdot t$
    $\mathbf{A}_f[b] = \mathbf{A}_f[b] \cdot (1 - r_i) + \mathbf{A}_f[b + 2^{\ell-i}] \cdot r_i$
    $\mathbf{A}_g[b] = \mathbf{A}_g[b] \cdot (1 - r_i) + \mathbf{A}_g[b + 2^{\ell-i}] \cdot r_i$
  for $t = 0, 1, 2$ **do**
    $a_{it} = \sum_{b \in \{0,1\}^{\ell-i}} f(r_1, \ldots, r_{i-1}, t, b) \cdot g(r_1, \ldots, r_{i-1}, t, b)$
return $\{a_1, \ldots, a_\ell\}$;

Next, we present an algorithm for $\mathcal{P}$ to run the sumcheck protocol on Equation 6 in linear time. We start with an algorithm to run sumcheck for the product of two multilinear polynomials in the literature, which we will use as a major building block.

Sumcheck for products of multilinear functions [46]. In [46], Thaler proposed a linear-time algorithm for the prover of the sumcheck protocol on the product of two multilinear polynomials f and g with ℓ variables (the algorithm runs in $O(2^\ell)$ time). We present the algorithm in Algorithm 1 and we have a lemma as follows:

Lemma 3.1. *Given multilinear functions f and g on ℓ variables and* bookkeeping tables $\boldsymbol{A}_f$ *and* $\boldsymbol{A}_g$ *for f and g respectively, the prover in Protocol 1 for $f \cdot g$ runs in $O(2^\ell)$ time. $\boldsymbol{A}_f = (f(0,\ldots,0), \ldots, f(1,\ldots,1))$ and $\boldsymbol{A}_g = (g(0,\ldots,0), \ldots, g(1,\ldots,1))$ are initialized with evaluations of f and g on the Boolean hypercube.*

Proof. In the i-th round during the sumcheck protocol, the prover runs $O(2^{\ell-i})$ steps to compute the required evaluations on f and g using bookkeeping tables of $\mathbf{A}_f$ and $\mathbf{A}_g$ respectively. $\mathcal{P}$ also updates two tables by fresh randomness in $O(2^{\ell-i})$ time as the size of the bookkeeping tables shrinks by half in each iteration. Then $\mathcal{P}$ computes three evaluations on $0, 1, 2$ to uniquely define the univariate polynomial of degree 2 in $O(2^{\ell-i})$ time given corresponding evaluations on f and g. Therefore, Algorithm 1 runs in $O(2^{\ell-1} + \ldots + 2^0) = O(2^\ell)$ time.

Linear-time sumcheck for Equation 6. The prover algorithm is similar to that proposed in [53]. The algorithm proceeds in two phases, one summing x and the other summing y. For the ease of presentation, consider the sumcheck on a particular class of equations:

$$\sum_{x,y \in \{0,1\}^\ell} \sum_{i=1}^{m} \tilde{\beta}((y_{k_i+1}, \ldots, y_\ell), \vec{1}) f_i(g,x,y_1,\ldots,y_{k_i}) t(x) s_i(y_1,\ldots,y_{k_i}), \quad (7)$$

for a fixed point $g \in \mathbb{F}^\ell$, where $t(x) : \mathbb{F}^\ell \to \mathbb{F}$ and $s_i(x) : \mathbb{F}^{k_i} \to \mathbb{F}$ are multilinear extensions of arrays $\mathbf{A}_t$ and $\mathbf{A}_{s_i}$, and all functions of $f_i(x) : \mathbb{F}^{2\ell+k_i} \to \mathbb{F}$ are multilinear extensions of sparse arrays with $O(2^\ell)$ nonzero elements in total. To simplify the equation,

Algorithm 2 $\mathbf{A}_{h_g} \leftarrow$ Initialize_Phase1($f_1, \ldots, f_m, s_1, \ldots, s_m, \mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_m}, g$)

Input: Multilinear $f_1, \ldots, f_m$ and $s_1, \ldots, s_m$, initial bookkeeping tables $\mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_m}$, random $g = g_1, \ldots, g_\ell$; We have $\sum_{i=1}^m |\mathbf{A}_{s_i}| = 2^\ell$.
Output: Bookkeeping table $\mathbf{A}_{h_g}$;

1: **procedure** $\mathbf{G} \leftarrow$ Precompute(g) // $\mathbf{G}$ *is an array of size* 2^ℓ.
2: Set $\mathbf{G}[0] = 1$
3: **for** $i = 0, \ldots, \ell - 1$ **do**
4: **for** $b \in \{0,1\}^i$ **do**
5: $\mathbf{G}[b, 0] = \mathbf{G}[b] \cdot (1 - g_{i+1})$
6: $\mathbf{G}[b, 1] = \mathbf{G}[b] \cdot g_{i+1}$
7: $\forall x \in \{0,1\}^\ell$, set $\mathbf{A}_{h_g}[x] = 0$
8: **for** every (z, x, y) such that $f_i'(z, x, y)$ is non-zero **do** // $f_i'(z, x, y) = \tilde{\beta}((y_{k_i+1}, \ldots, y_\ell), \vec{1}) f_i(z, x, y_1, \ldots, y_{k_i})$
9: $\mathbf{A}_{h_g}[x] = \mathbf{A}_{h_g}[x] + \mathbf{G}[z] \cdot f_i'(z, x, y) \cdot \mathbf{A}_{s_i}[y_1, \ldots, y_{k_i}]$
10: **return** $\mathbf{A}_{h_g}$;

let $f_i'(g, x, y) = \tilde{\beta}((y_{k_i+1}, \ldots, y_\ell), \vec{1}) f_i(g, x, y_1, \ldots, y_{k_i})$. In addition, we require that $\sum_{i=1}^m 2^{k_i} = 2^\ell$. It is not hard to see that Equation 6 satisfies these properties, as there are at most S_i left input gates and S_i right input gates connected to layer i in the circuit C. If we set $\ell = s_i = \lceil \log S_i \rceil$, we have $\sum_{i=1}^m 2^{k_i} = O(S_i)$ in Equation 6.

We use the same intuition in [53] of dividing the sumcheck process into two phases, one is for x and the other is for y. We rewrite Equation 7 as follows $\sum_{x \in \{0,1\}^\ell} t(x) h_g(x)$, where

$$h_g(x) = \sum_{y \in \{0,1\}^\ell} \sum_{i=1}^m f_i'(g, x, y) s_i(y_1, \ldots, y_{k_i}) \tag{8}$$

Phase one. With the formula above, in the first ℓ rounds, the prover and the verifier are running exactly a sumcheck on the product of two multilinear polynomials $t(x) \cdot h_g(x)$. To compute the sumcheck messages for the first ℓ rounds, given their bookkeeping tables, it takes $O(2^\ell)$ time by Lemma 3.1. It remains to show how to initialize the bookkeeping tables in linear time.

Initializing the bookkeeping tables:

Initializing the bookkeeping table for t in $O(2^\ell)$ time is trivial, since t is a multilinear extension of an array and therefore the evaluations on the Boolean hypercube are known. Initializing the bookkeeping table for h_g in $O(2^\ell)$ time is more challenging, but we can take advantage of the sparsity of f_i'.

Lemma 3.2. *Let $\mathcal{N}_x$ be the set of $(z, y) \in \{0,1\}^{2\ell}$ such that $f_i'(z, x, y)$ is non-zero for some i. Then for all $x \in \{0,1\}^\ell$, $h_g(x) = \sum_{(z,y) \in \mathcal{N}_x} \tilde{\beta}(g, z) \cdot \sum_{i=1}^m f_i'(z, x, y) \cdot s_i(y_1, \ldots, y_{k_i})$.*

Proof. Since f_i is a multilinear extension, as shown in [46], we have $f_i'(g, x, y) = \sum_{z \in \{0,1\}^\ell} \tilde{\beta}(g, z)\, f_i'(z, x, y)$. Therefore,

$$\begin{aligned} h_g(x) &= \sum_{y, z \in \{0,1\}^\ell} \tilde{\beta}(g, z) \cdot \sum_{i=1}^m f_i'(z, x, y) \cdot s_i(y_1, \ldots, y_{k_i}) \\ &= \sum_{(z,y) \in \mathcal{N}_x} \tilde{\beta}(g, z) \cdot \sum_{i=1}^m f_i'(z, x, y) \cdot s_i(y_1, \ldots, y_{k_i}) \end{aligned}$$

Lemma 3.3. *$\mathbf{A}_{h_g}$ can be initialized in time $O(2^\ell)$.*

Proof. As f_i' is sparse, $\sum_{x \in \{0,1\}^\ell} |\mathcal{N}_x| = O(2^\ell)$. From Lemma 3.2, given the evaluations of $\tilde{\beta}(g, z)$ for all $z \in \{0,1\}^\ell$, the prover can iterate through all $(z, y) \in \mathcal{N}_x$ for all x to compute $\mathbf{A}_{h_g}$. The full algorithm is presented in Algorithm 2. Since each s_i is the multilinear extension of an array, its evaluations on the Boolean hypercube are known. Therefore, we use $\mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_m}$ as the input of Algorithm 2. $\sum_{i=1}^m |\mathbf{A}_{s_i}| = 2^\ell$ as $\sum_{i=1}^m 2^{k_i} = 2^\ell$.

Procedure Precompute(g) is to evaluate $\mathbf{G}[z] = \tilde{\beta}(g, z) = \prod_{i=1}^\ell ((1 - g_i)(1 - z_i) + g_i z_i))$ for $z \in \{0,1\}^\ell$. By the closed-form of $\tilde{\beta}(g, z)$, the procedure iterates each bit of z, and multiples $1 - g_i$ for $z_i = 0$ and multiples g_i for $z_i = 1$. In this way, the size of $\mathbf{G}$ doubles in each iteration, and the total complexity is $O(2^\ell)$.

Step 8-9 computes $h_g(x)$ using Lemma 3.2. When f_i' is represented as a map of $((z, x, y), f_i'(z, x, y))$ for non-zero values, the complexity of these steps is $O(2^\ell)$ since $\sum_{x \in \{0,1\}^\ell} |\mathcal{N}_x| = O(2^\ell)$.

In Protocol 4, the map above is exactly the representation of a gate in the circuit, where z, x, y are labels of the gate, its left input and its right input, and $f_i'(z, x, y) = 1$.

With the bookkeeping tables, the prover runs Algorithm 1 for the product of multilinear polynomials and the total complexity for phase one is $O(2^\ell)$.

Phase two. At this point, the variable x is set to random numbers $u \in \mathbb{F}^\ell$. In the second phase, the equation to sum on becomes

$$t(u) \sum_{y \in \{0,1\}^\ell} \sum_{i=1}^m f_i'(g, u, y) s_i(y_1, \ldots, y_{k_i})$$

Note here that $t(u)$ is merely a constant which the prover already computed in phase one. For the part behind the summation symbol on y, it has m products of two multilinear functions to sum. If we naively apply Algorithm 1 to each product, the prover runs in $O(m \cdot 2^\ell)$ time instead of only $O(2^\ell)$. Fortunately, we observe that the total size of m bookkeeping tables is $O(2^\ell)$ as $\sum_{i=1}^m 2^{k_i} = 2^\ell$, which reduces the number of operations on all bookkeeping tables to $O(2^\ell)$ and removes the m factor in the complexity. To achieve the linear prover time, we generalize Lemma 3.1 to Lemma 3.4 for the summation of multiple products of multilinear functions.

Lemma 3.4. *Suppose we have $2m$ multilinear polynomials $f_1, f_2, \ldots, f_m$ and $g_1, g_2, \ldots, g_m$. Both g_i and f_i have k_i variables. Without loss of generality, suppose $\ell \geq k_m \geq k_{m-1} \geq k_1$. If $\sum_{i=1}^m 2^{k_i} = 2^\ell$, given the* bookkeeping *tables $\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m}$ for $f_1, \ldots, f_m$ and $\mathbf{A}_{g_1}, \cdot, \mathbf{A}_{g_m}$ for $g_1, \ldots, g_m$, the prover in Protocol 1 for $\sum_{y \in \{0,1\}^{k_i}} \sum_{i=1}^m f_i(y) \cdot g_i(y) = \sum_{y \in \{0,1\}^\ell} \sum_{i=1}^m \tilde{\beta}((y_{k_i+1}, \ldots, y_\ell), \vec{1}) \cdot f_i(y_1, \ldots, y_{k_i}) \cdot g_i(y_1, \ldots, y_{k_i})$ runs in $O(2^\ell)$ time.*

Proof. We present Algorithm 3 for the prover in the sumcheck. In the j-th round during the sumcheck protocol, for the product of $f_q \cdot g_q$ such that $j \leq k_q$, the prover computes required evaluatons on f_q and g_q using $\mathbf{A}_{f_q}$ and $\mathbf{A}_{g_q}$ and updates $\mathbf{A}_{f_q}$ and $\mathbf{A}_{g_q}$ by fresh randomness in $O(2^{k_q - j})$ time. The process is extactly the same as step 2-7 in Algorithm 1 except for the table size of 2^{k_q}. For the product of $f_q \cdot g_q$ such that $j > k_q$, the bookkeeping tables of f_q and g_q already shrinked to one point of $f_i(r_1, \ldots, r_{k_i}) \cdot g_i(r_1, \ldots, r_{k_i})$ in the k_q-th round. The evaluation does not change in following rounds, thus $\mathcal{P}$ can compute related messages of $f_q \cdot g_q$ and update $f_q \cdot g_q$ by fresh randomness in a single step. This concludes that

Algorithm 3 $\{a_1, \ldots, a_\ell\} \leftarrow$ SumCheckProduct2 $(f_1, \mathbf{A}_{f_1}, g_1, \mathbf{A}_{g_1}, \ldots, f_m, \mathbf{A}_{f_m}, g_m, \mathbf{A}_{g_m}, r_1, \ldots, r_\ell)$

Input: Multilinear f_i and g_i, initial bookkeeping tables $\mathbf{A}_{f_i}$ and $\mathbf{A}_{g_i}$ for $i = 1$ to m, random $r_1, \ldots, r_\ell$; we have $\sum_{i=1}^m |\mathbf{A}_{f_i}| = \sum_{i=1}^m |\mathbf{A}_{g_i}| = 2^\ell$.
Output: ℓ sumcheck messages for $\sum_{y\in\{0,1\}^\ell} \sum_{i=1}^m \tilde{\beta}((y_{k_i+1}, \ldots, y_\ell), \vec{1}) f_i(y_1, \ldots, y_{k_i}) \cdot g_i(y_1, \ldots, y_{k_i})$. Each message a_i consists of 3 elements (a_{i0}, a_{i1}, a_{i2});

1: **for** $i = 1, \ldots, m$ **do**
2: **for** $j = k_i + 1, \ldots, k_{i+1}$ **do** *// Suppose* $k_0 = 0 < k_1 \leq \ldots \leq k_m \leq k_{m+1} = \ell$
3: **for** $q = i + 1, \ldots, m$ **do**
4: **for** $b \in \{0, 1\}^{k_q - j}$ **do**
5: **for** $t = 0, 1, 2$ **do**
6: Let $f_q(r_1, \ldots, r_{j-1}, t, b) = \mathbf{A}_{f_q}[b]\cdot(1-t)+\mathbf{A}_{f_q}[b+2^{k_q-j}] \cdot t$
7: Let $g_q(r_1, \ldots, r_{j-1}, t, b) = \mathbf{A}_{g_q}[b] \cdot (1 - t) + \mathbf{A}_{g_q}[b + 2^{k_q-j}] \cdot t$
8: $\mathbf{A}_{f_q}[b] = \mathbf{A}_{f_q}[b] \cdot (1 - r_j) + \mathbf{A}_{f_q}[b + 2^{k_q-j}] \cdot r_j$
9: $\mathbf{A}_{g_q}[b] = \mathbf{A}_{g_q}[b] \cdot (1 - r_j) + \mathbf{A}_{g_q}[b + 2^{k_q-j}] \cdot r_j$ *// step 3-9 has the same procedure as step 2-7 in Algorithm 1*
10: **for** $t \in \{0, 1, 2\}$ **do**
11: $a_{jt} = \sum_{q=1}^{i} t \cdot f_q(r_1, \ldots, r_{k_q}) \cdot g_q(r_1, \ldots, r_{k_q}) + \sum_{q=i+1}^{m} \sum_{b\in\{0,1\}^{k_q-j}} f_q(r_1, \ldots, r_{j-1}, t, b) \cdot g_q(r_1, \ldots, r_{j-1}, t, b)$
12: **for** $q = 1, \ldots, i$ **do**
13: $f_q(r_1, \ldots, r_{k_q}) \cdot g_q(r_1, \ldots, r_{k_q}) = r_j \cdot f_q(r_1, \ldots, r_{k_q}) \cdot g_q(r_1, \ldots, r_{k_q})$
14: **return** $\{a_1, \ldots, a_\ell\}$;

$f_q \cdot g_q$ consumes $O(2^{k_q})$ operations during the first k_q rounds as shown in Lemma 3.1 and consumes $O(\ell - k_q)$ operations in the next $\ell - k_q$ rounds. Therefore, $\mathcal{P}$ runs in $O(\sum_{q=1}^m 2^{k_q} + \ell - k_q) = O(2^\ell)$ time for Algorithm 3 assuming $O(m\ell) \ll O(2^\ell)$ in most cases. Furthermore, $\mathcal{P}$ can also aggregate $f_1 \cdot g_1$ to $f_q \cdot g_q$ after round k_q to remove the term of $O(m\ell)$ in the complexity. We omit the optimization in Algorithm 3.

The sumcheck polynomial for phase two has the same form in Lemma 3.4. To compute the sumcheck messages for the last ℓ rounds, given their bookkeeping tables, this will take $O(2^\ell)$ time by Lemma 3.4. We now show how to initialize the bookkeeping tables in linear time.

Initializing the bookkeeping tables:

Initializing the bookkeeping table for each s_i in $O(2^{k_i})$ time is trivial, since each s_i is a multilinear extension of an array and therefore the evaluations on the hypercube are known. We also know $\sum_{i=1}^m 2^{k_i} = O(2^\ell)$. It remains to initialize bookkeeping tables for all f_i in $O(2^\ell)$ time. Similar to phase one, we can leverage the sparsity of f_i' and we have the lemma as follows:

Lemma 3.5. *Let $\mathcal{N}_y$ be the set of $(z, x) \in \{0, 1\}^{2\ell}$ such that $f_i'(z, x, y)$ is non-zero for some i. Then for all $y \in \{0, 1\}^\ell$, it is $f_i'(g, u, y) = \sum_{(z,x)\in\mathcal{N}_y} \tilde{\beta}(g, z)\tilde{\beta}(u, x) f_i'(z, x, y)$*

Algorithm 4 $\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m} \leftarrow$ Initialize_Phase2 $(f_1, \ldots, f_m, g, u)$

Input: Multilinear $f_1, \ldots, f_m$, random $g = g_1, \ldots, g_m$ and $u = u_1, \ldots, u_\ell$;
Output: Bookkeeping tables $\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m}$;

1: $\mathbf{G} \leftarrow$ Precompute(g) *// Invoke* Precompute *procedure in Algorithm 2*
2: $\mathbf{U} \leftarrow$ Precompute(u) *// Invoke* Precompute *procedure in Algorithm 2*
3: $\forall y \in \{0, 1\}^\ell$, set $\mathbf{A}_{f_i}[y_1, \ldots, y_{k_i}] = 0$ for all i
4: **for** every (z, x, y) such that $f_i'(z, x, y)$ is non-zero **do**
5: $\mathbf{A}_{f_i}[y_1, \ldots, y_{k_i}] = \mathbf{A}_{f_i}[y_1, \ldots, y_{k_i}] + \mathbf{G}[z] \cdot \mathbf{U}[x] \cdot f_i'(z, x, y)$
6: **return** $\mathbf{A}_{f_1}, \mathbf{A}_{f_2}, \ldots, \mathbf{A}_{f_m}$;

Proof. Lemma 3.5 is a generalization of Lemma 3.2. Since f_i is a multilinear extension, as shown in [46], we have,

$$f_i'(g, u, y) = \sum_{z,x\in\{0,1\}^\ell} \tilde{\beta}(g, z)\tilde{\beta}(u, x) f_i'(z, x, y) = \sum_{(z,x)\in\mathcal{N}_y} \tilde{\beta}(g, z)\tilde{\beta}(u, x) f_i'(z, x, y)$$

Lemma 3.6. *$\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m}$ can be initialized in time $O(2^\ell)$.*

Proof. As f_i is sparse, $\sum_{y\in\{0,1\}^\ell} |\mathcal{N}_y| = O(2^\ell)$. From Lemma 3.2, given the evaluations of $\tilde{\beta}(g, z)$ and $\tilde{\beta}(u, x)$ for all $z, x \in \{0, 1\}^\ell$, the prover can iterate all $(z, x) \in \mathcal{N}_x$ for all y to compute $\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m}$. The full algorithm is presented in Algorithm 4.

$\mathcal{P}$ runs procedure Precompute(g) and Precompute(u) in $O(2^\ell)$ time as we have shown in the proof of Lemma 3.3. Step 4-5 computes $f_i(y_1, \ldots, y_{k_i})$ using Lemma 3.5. It takes $O(2^\ell)$ time as $\sum_{y\in\{0,1\}^\ell} |\mathcal{N}_y| = O(2^\ell)$. Therefore, $\mathcal{P}$ runs in $O(2^\ell)$ time for Algorithm 4.

With the bookkeeping tables, the prover runs SumCheckProduct2 $(f_1, \mathbf{A}_{f_1}, g_1, \mathbf{A}_{g_1}, \ldots, f_m, \mathbf{A}_{f_m}, g_m, \mathbf{A}_{g_m}, r_1, \ldots, r_\ell)$ in Algorithm 3 and the total complexity for phase two is $O(2^\ell)$.

Combining phase one and phase two, we know that $\mathcal{P}$ runs in $O(|C|)$ time for the sumcheck protocol on Equation 7.

Step one with linear prover time. Finally, the sumcheck protocol for Equation 6 can be decomposed into several instances that have the form of Equation 7. The term

$$\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} \sum_{j=i+1}^{d} \tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) \tilde{mult}_{i+1,j}(g, x, y_1, \ldots, y_{s_{i,j}}) \tilde{V}_{i,i+1}(x) \tilde{V}_{i,j}(y_1, \ldots, y_{s_{i,j}})$$

is exactly the same as Equation 7. The term

$$\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} \sum_{j=i+1}^{d} \tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) \tilde{add}_{i+1,j}(g, x, y_1, \ldots, y_{s_{i,j}})(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,j}(y_1, \ldots, y_{s_{i,j}}))$$

can be rewritten as the sum of

$$\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} \sum_{j=i+1}^{d} \tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) \tilde{add}_{i+1,j}(z, x, y_1, \ldots, y_{s_{i,j}}) \tilde{V}_{i,i+1}(x)$$

and

$$\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} \sum_{j=i+1}^{d} \tilde{\beta}((y_{s_{i,j}+1}, \ldots, y_{s_{i,i+1}}), \vec{1}) \tilde{add}_{i+1,j}(z, x, y_1, \ldots, y_{s_{i,j}}) \tilde{V}_{i,j}(y_1, \ldots, y_{s_{i,j}}).$$

The first sum is the same as Equation 7 with $s_i(x) = 1$, and the second sum is the same as Equation 7 with $t(x) = 1$. The complexity for both cases remains linear. Due to linearity of the sumcheck protocol, the prover can execute these 3 instances simultaneously in every round, sum up messages and send them to the verifier.

Verifier time and proof size for all sumcheck protocols on Equation 6. The verifier time for all sumcheck protocols on Equation 6 is the same as Protocol 2. $\mathcal{V}$ still runs in $O(d \log |C|)$ time to verify all sumcheck statements based on Equation 6. The proof size is also $O(d \log |C|)$. Note that this excludes the claims of $\tilde{V}_{i,j}$ at random points at the end of the sumcheck protocol in each layer, and we will count them in the next section combining these claims.

3.3 Combining Multiple Claims in Linear Time

By executing the sumcheck protocol on Equation 6, $\mathcal{P}$ and $\mathcal{V}$ reduce one claim about a layer to multiple claims about the layers above. As we explained in Section 3.1, when reaching layer i, $\mathcal{V}$ has received multiple evaluations about this layer and combining these evaluations using a random linear combination would introduce an overhead on the prover. Even worse, with the refined sumcheck on Equation 6 in Section 3.2, now the verifier has received multiple evaluations of *different* multilinear extensions defined by subsets of gates in layer i used by different layers below. Now even combining these evaluations becomes challenging, let alone reducing the overhead of the prover.

In this section, we propose two approaches that combine these evaluations to a single evaluation of the multilinear extension of the entire layer i and incur a linear prover time in the circuit size.

Combining multiple claims by an arithmetic circuit. The key idea of the first approach is that instead of trying to come up with a complicated protocol to do the combination, we simply model the computation as an arithmetic circuit! The circuit takes the values $\mathbf{V}_i$ of the entire layer i as the input. In addition, it also takes the randomness to compute these evaluations of subsets from the verifier. At this point, these randomness are already chosen by the verifier and can merely be viewed as constants known both to $\mathcal{V}$ and $\mathcal{P}$. The circuit then selects multiple subsets from $\mathbf{V}_i$ according to the wiring of the circuit (i.e., gates used by layer $j < i$ from layer i), arrange them in the pre-defined order. The circuit then computes the multilinear extensions of these subsets, and evaluates them on the corresponding points from the input. The structure of the circuit C_i is given in Figure 2.

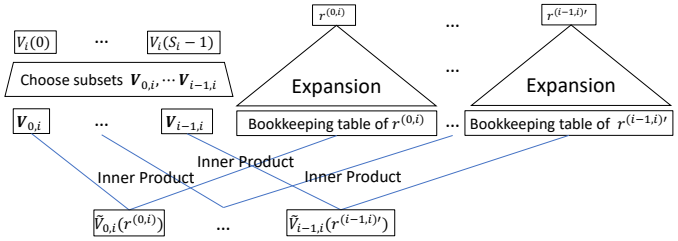


Figure 2: Circuit C_i computing $\tilde{V}_{0,i}(r^{(0,i)}), \cdots, \tilde{V}_{i-1,i}(r^{(i-1,i)'})$

The output of the circuit is exactly the multiple evaluations of the multilinear extensions, which are known to the verifier. The verifier then executes the original GKR protocol for layered arithmetic circuits (Protocol 2) on this circuit, which reduces the output to a single evaluation of the multilinear extension of the input. This can further be expressed as the evaluation of the multilinear extension of $\mathbf{V}_i$, together with the multilinear extension of all the randomnesses used to compute the output. As the latter is known to $\mathcal{V}$, $\mathcal{V}$ can compute it locally. In this way, $\mathcal{P}$ and $\mathcal{V}$ reduce multiple claims about subsets of layer i to one claim about $\tilde{V}_i$ by C_i.

Another tricky part is that the size of the circuit is not optimal if implemented naively. As shown in Figure 2, the circuit expands the randomness to the bookkeeping tables exactly as described in Algorithm 1, which has logarithmic layers $\log S_i$. If the circuit also takes $\mathbf{V}_i$ as input at the same layer as the randomness, $\mathbf{V}_i$ has to be relayed by logarithmic layers and the size of the circuit is $O(S_i \log S_i)$. Instead, we feed $\mathbf{V}_i$ as input to one layer above the bookkeeping tables, as shown on the left side of Figure 2. The circuit selects multiple subsets out of it in one layer, and then computes the inner product between a subset and its corresponding bookkeeping table, which gives the evaluation of its multilinear extension. Now the size of the circuit is linear to the total size of all the subsets. The GKR protocol can support inputs from different layers with such a structure, as proposed in [56]. We give the formal protocol in Protocol 3 and the efficiency analysis in Appendix B.

Combining multiple claims by a sumcheck protocol. Though the prover time of the first method is optimal asymptotically, the overhead in practice is still relatively high. As we will show in Section 5, the cost per gate is around 5× slower than that of the original GKR protocol on layered circuits. In addition, it introduces an overhead of $O(\log |C|)$ on the proof size and the verifier time. Therefore, inspired by the design of the circuit in Figure 2, we propose the second method to combine multiple claims through a single sumcheck protocol.

The key idea is to define a function to connect the gate in V_i with the same gate in a subset $V_{k,i}$. Formally speaking, we define $C_{k,i}(z,x) : \{0,1\}^{s_{k,i}} \times \{0,1\}^{s_i} \to \mathbb{F}$ such that it takes two gate labels, one in the subset $V_{k,i}$ and the other in the entire layer V_i, and $C_{k,i}(z,x) = 1$ if the gate z in $V_{k,i}$ is exactly the gate x in V_i. Otherwise $C_{k,i}(z,x) = 0$. The function $C_{k,i}$ serves exactly the same purpose as the circuit in Figure 2 selecting subsets from $\mathbf{V}_i$.

With the definition of $C_{k,i}$, given $\tilde{V}_{0,i}(r^{(0,i)}), \ldots, \tilde{V}_{i-1,i}(r^{(i-1,i)})$, $\tilde{V}_{i-1,i}(r^{(i-1,i)'})$, $\mathcal{V}$ can combine them through a random linear combination with randomness of $\alpha_{0,i}, \ldots, \alpha_{i-1,i}, \alpha'_{i-1,i}$. Then we have

$$
\begin{aligned}
&\sum_{k=0}^{i-1} \alpha_{k,i} \tilde{V}_{k,i}(r^{(k,i)}) + \alpha'_{i-1,i} \tilde{V}_{i-1,i}(r^{(i-1,i)'}) \\
&= \sum_{k=0}^{i-1} \left(\alpha_{k,i} \sum_{x \in \{0,1\}^{s_i}} \tilde{C}_{k,i}(r^{(k,i)}, x) \tilde{V}_i(x) \right) + \\
&\quad \alpha'_{i-1,i} \sum_{x \in \{0,1\}^{s_i}} \tilde{C}_{i-1,i}(r^{(i-1,i)'}, x) \tilde{V}_i(x) \\
&= \sum_{x \in \{0,1\}^{s_i}} \tilde{V}_i(x) \left(\sum_{k=0}^{i-1} \alpha_{k,i} \tilde{C}_{k,i}(r^{(k,i)}, x) + \alpha'_{i-1,i} \tilde{C}_{i-1,i}(r^{(i-1,i)'}, x) \right) \\
&= \sum\nolimits_{x \in \{0,1\}^{s_i}} \tilde{V}_i(x) g_i(x), \qquad (9)
\end{aligned}
$$

Algorithm 5 $\mathbf{A}_{g_i} \leftarrow \text{Initialize}(r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'})$

Input: $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$;
Output: $\mathbf{A}_{g_i}$;

1: $\forall x \in \{0,1\}^{s_i}$, set $\mathbf{A}_{g_i}[x] = 0$
2: **for** $k = 0, \cdots, i-1$ **do**
3: $\quad \mathbf{G} \leftarrow \text{Precompute}(r^{(k,i)})$
4: $\quad$ **for** $t \in \{0,1\}^{s_{k,i}}$ such that $C_{k,i}(t,x) = 1$ **do**
5: $\quad\quad \mathbf{A}_{g_i}[x] = \mathbf{A}_{g_i}[x] + \alpha_{k,i} \cdot \mathbf{G}[t]$
6: $\mathbf{G} \leftarrow \text{Precompute}(r^{(i-1,i)'})$
7: **for** $t \in \{0,1\}^{s_{i-1,i}}$ such that $C_{i-1,i}(t,x) = 1$ **do**
8: $\quad \mathbf{A}_{g_i}[x] = \mathbf{A}_{g_i}[x] + \alpha'_{i-1,i} \cdot \mathbf{G}[t]$
9: **return** $\mathbf{A}_{g_i}$;

where $\tilde{C}_{k,i}$ is the multilinear extension of $C_{k,i}$ and $\tilde{C}_{k,i}(r^{(k,i)}, x) = \sum_{z\in\{0,1\}^{s_{k,i}}} \tilde{\beta}(r^{(k,i)}, z) C_{k,i}(z,x)$.

We define $g_i(x) = \sum_{k=0}^{i-1} \alpha_{k,i}\tilde{C}_{k,i}(r^{(k,i)}, x) + \alpha'_{i-1,i}\tilde{C}_{i-1,i}(r^{(i-1,i)'}, x)$. As $g_i(x)$ only depends on the structure of the circuit, $\mathcal{V}$ can compute $g_i(r^{(i)})$ herself given $r^{(i)}$. $\mathcal{P}$ and $\mathcal{V}$ can execute a sumcheck protocol on Equation 9, which reduces multiple claims of subsets to a single evaluation of $\tilde{V}_i(r^{(i)})$ for the randomness of $r^{(i)}$.

It remains to show that the sumcheck can be executed by the prover in linear time. Recall that given the bookkeeping tables of two multilinear polynomials, the prover can run the sumcheck protocol in linear time using Algorithm 1. In the equation above, the bookkeeping table $\mathbf{A}_{\tilde{V}_i}$ for $\tilde{V}_i$ is already known by the prover as the gate values in layer i. We further describe a linear time algorithm to initialize the bookkeeping table $\mathbf{A}_{g_i}$ for $g_i(x)$ in Algorithm 5.

Lemma 3.7. *$\mathbf{A}_{g_i}$ can be initialized in $O(S_{0,i} + \cdots + S_{i-1,i})$ time.*

Proof. Algorithm 5 initializes $\mathbf{A}_{g_i}$ in $O(S_{0,i} + \cdots + S_{i-1,i})$ time. For each random vector $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$, Algorithm 5 invokes procedure Precompute to compute the corresponding array $\mathbf{G}$ with size of $S_{0,i}, \ldots, S_{i-1,i}, S_{i-1,i}$. Hence $\mathcal{P}$ costs $O(S_{0,i} + \ldots + S_{i-1,i} + S_{i-1,i})$ time for all revocations as shown in the proof of Lemma 3.3. Given all of these arrays, if there exists a pair of (t,x) such that $C_{k,i}(t,x) = 1$, $\mathbf{G}[t]$ contributes to the bookkeeping table on the cell of x. There are at most $S_{0,i} + \ldots + S_{i-1,i} + S_{i-1,i}$ entries such that $C_{k,i}(t,x) = 1$ since there are at most $S_{0,i} + \ldots + S_{i-1,i} + S_{i-1,i}$ input wires from layer i. Therefore, the running time of Algorithm 5 is $O(S_{0,i} + \cdots + S_{i-1,i})$.

Efficiency. Next we analyze the efficiency of the second scheme to combine multiple claims to one claim. The prover costs $O(S_1 + \ldots + S_d) = O(|C|)$ to compute all bookkeeping tables of $A_{\tilde{V}_i}$. By Lemma 3.7, the prover runs Algorithm 5 to compute all bookkeeping tables of $\mathcal{A}_{g_i}$ in $O(\sum_{i=1}^{d}\sum_{k=0}^{i-1} S_{k,i}) = O(|C|)$ time. By Lemma 3.1, the prover runs the sumcheck protocol on Equation 9 for layer 1 to layer d in $O(\sum_{i=1}^{d} S_i) = O(|C|)$ time. So the total prover time of the second scheme is also linear in the circuit size.

By the efficiency of the sumcheck protocol in Protocol 1, it takes $O(\log S_i)$ for the verifier to validate the sumcheck protocol on Equation 9 in round i. She also needs to generate $i+1$ random numbers and computes $g_i(r^{(i)})$ in round i. Suppose $\mathcal{V}$ costs T_i to compute $g_i(r^{(i)})$ and $T = T_1 + \ldots + T_d$, the total verifier time is $O(\log S_1 + \ldots + \log S_d) + O(2 + \ldots + d + 1) + T_1 + \ldots + T_d = O(d \log |C| + d^2 + T)$. The total proof size is $O(d \log C + d^2)$. Finally, by a similar analysis to the prover time, the term $O(d^2)$ in the complexity is always bounded by $O(|C|)$. This is because in order for the prover to send a claim about $\tilde{V}_{i,j}$, there has to be a gate in layer i connecting to layer j, thus the number of claims cannot be more than $2|C|$. Therefore, the proof size of our protocol is $\min\{O(d \log C + d^2), O(|C|)\}$ and the verifier time is $\min\{O(d \log |C| + d^2 + T), O(|C|)\}$.

3.4 The Full Protocol for General Circuits

Combining the first step and the sumcheck scheme of the second step together, we give the full protocol of the generalized GKR for arbitrary arithmetic circuits in Protocol 4 in Appendix C. As the prover time, proof size and the verifier time of the second method to combine multiple points are all better than those of the first method, we state the protocol and the theorem using the second method. We provide the formal proof of Theorem 3.8 in Appendix C.

Theorem 3.8. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a d-depth general arithmetic circuit. Protocol 4 is an interactive proof for the function computed by C with soundness $O(d \log |C|/|\mathbb{F}|)$. The running time of the prover $\mathcal{P}$ is $O(|C|)$. The proof size is $\min\{O(d \log C + d^2), O(|C|)\}$. Let the time to evaluate all $\tilde{add}_{i,j}$ and $\tilde{mult}_{i,j}$ at random points be T', the time to evaluate $g_i(r^{(i)})$ be T_i in Equation 9, and $T = T_1 + \ldots + T_d$, the running time of $\mathcal{V}$ is $\min\{O(n + d \log |C| + d^2 + T + T'), O(|C|)\}$. If in addition $d = \text{polylog}(|C|)$, T and T' are in $\text{polylog}(|C|)$ time in Theorem 3.8, Protocol 4 is an interactive proof with succinct proof size and verifier time.*

4 OUR ZERO KNOWLEDGE ARGUMENTS

In this section, we build a new zero knowledge argument protocol for general arithmetic circuits based on Protocol 4. The construction follows the same ideas proposed in [23, 53]. In particular, as proposed in [56], we combine the GKR protocol with a (zero knowledge) polynomial commitment scheme on the witness to build an argument scheme. In order to achieve zero knowledge, we apply the zero knowledge sumcheck protocol [23, 53] on Equation 6 and 9 to eliminate the leakage during the sumcheck. We then use the low degree extensions instead of multilinear extensions of V_i and $V_{i,j}$ so that their evaluations sent from the prover to the verifier do not leak information about the values in the circuit. The only difference is that for the values V_i in each layer i of the circuit, the verifier receives multiple claims, one for each of the subsets $V_{i,j}$, instead of two claims about V_i in the original GKR protocol. Thus, we use the low degree extensions of both V_i and $V_{i,j}$ with a different random masking polynomial for each. In this way, these claims leak no information about the values.

For completeness, we present the formal definitions and protocols in Appendix D. As the second approach to combine multiple claims in Section 3.3 is better on all aspects, we focus on building zero knowledge arguments using the second approach. The first approach can also be lifted to a zero knowledge argument with similar ideas by applying a zero knowledge GKR protocol on circuit C_i in Figure 2. We state the theorem of our zero knowledge argument here and give the proof in Appendix D.4.

Theorem 4.1. *For an input size n and a finite field $\mathbb{F}$, let $\mathcal{C}_{\mathbb{F}}$ represent the set of general arithmetic circuits of depth d on $\mathbb{F}$, then there*

exists a zero knowledge argument for the relation

$$\mathcal{R} = \{(C, x; w) : C \in \mathcal{C}_{\mathbb{F}} \wedge |x| + |w| \leq n \wedge C(x; w) = 1\},$$

as defined in Definition D.1. Moreover, using the polynomial commitment scheme (Definition D.2) in [55], for every $(C, x; w) \in \mathcal{R}$*, the running time of* $\mathcal{P}$ *is* $O(|C| + n \log n)$*. The running time of* $\mathcal{V}$ *is* $\min\{O(|x| + \log^2 n + d \log |C| + d^2 + T''), O(|C|)\}$*, where* T'' *is the total time to compute all functions of* $\tilde{add}$ *and* $\tilde{mult}$ *and all functions of* $g_i(r^{(i)})$ *in the second step. The total proof size is* $\min\{O(d \log |C| + d^2 + \log^2 n), O(|C|)\}$*. In case* d *is* $\text{polylog}(|C|)$ *and* T'' *is also* $\text{polylog}(|C|)$*, the protocol is a succinct argument with succinct verifier time.*

5 IMPLEMENTATIONS AND EVALUATIONS

We fully implement our new interactive proof protocols for general arithmetic circuits and use them to build a zero knowledge argument system for general arithmetic circuits. We name our new system Virgo++. The implementation is in C++. There are around 1900 lines of code for Protocol 4 and 1600 lines for building the arithmetic circuit to combine multiple evaluations into one (Protocol 3). We implement two variants of combining multiple claims to one claim in step 3(b) of Protocol 4 as described in Section 3.3. One is building the arithmetic circuit to make the reduction as shown in Figure 2 and the other is running the sumcheck protocol on Equation 9. Our protocols work on any finite field, and we choose the extension field $\mathbb{F}_{p^2}$ for the Mersenne prime $p = 2^{61} - 1$. This is the same as in [55], and we choose it so that our interactive proof protocols can be compatible with the polynomial commitments in [55] to build zero knowledge arguments. The choice of the finite field does not affect our comparison to the original GKR protocol in the next Section. Our protocols provide 100+ bits of security. We plan to make our implementation open-source.

Hardware. We ran all the experiments on an AWS EC2 m5d.4xlarge instance with Intel(R) Xeon(R) Platinum 8175M CPU with 2.50GHz, 8 cores and 64GB of RAM. Our current implementation is not parallelized and we only utilize a single CPU core in the experiments.

5.1 Comparing to the GKR Protocol

In this section, we compare the performance of our new protocols with the original GKR protocol. For a fair comparison, we re-implement the GKR protocol for layered arithmetic circuits with the same field and libraries in C++. We generate random general circuits with depth $d = 50$ and $d = 75$. We vary the number of gates in each layer from 2^9 to 2^{13}. Our schemes can easily go beyond 2^{13}, but the original GKR protocol on the corresponding layered circuits runs out of memory on our machine. We randomly sample the type of each gate, input value and the wiring patterns. We execute our new protocols directly on these general circuits. We refer the one using the arithmetic circuit to combine multiple claims to one claim in Protocol 3 as scheme 1 and the one using the sumcheck protocol on Equation 9 to combine multiple claims for step 3(b) in Protocol 4 as scheme 2. We then transform the general circuits to layered circuits by relaying necessary values layer by layer, and execute the original GKR protocol on the layered circuits. We report the prover time, verifier time and proof size in Table 1.

First, when we transform the general circuits to layered circuits, the size of the circuit increases by 13× for $d = 50$ and by 19× for $d = 75$. This roughly agrees with the blowup of $O(d|C|)$ and justifies the high overhead of transforming general circuits to layered circuits. As shown in Table 1, when the depth is 50, the prover time of our scheme 1 is faster by 2-4× than the original GKR protocol, while our scheme 2 is faster by 9-10×. When the depth is 75, the speedup increases to 3-5× for scheme 1 and 12-13× for scheme 2. Finally, the prover time in all schemes grows linearly with the size of the circuit, and is very efficient in practice. The cost per gate in scheme 2 is only 0.49μs.

To further justify the improvement, the prover of the original GKR protocol for layered circuits takes around 21 field multiplications per gate. In the implementation of our new protocols, the cost per gate of the prover is around 120 field multiplications for scheme 1 and around 27 field multiplications for scheme 2. The average cost per gate of our scheme 2 is only 1.29× of the original GKR protocol. In other words, as long as the layered circuit has 22% or more relay gates, it is faster to remove those relay gates and run our new protocol of scheme 2 on the corresponding general circuit. The speedup in our experiments above matches the analysis here.

Our protocols introduce an overhead on the proof size compared to the original GKR protocol. In particular, the proof size of our first scheme is 3-5× larger than the GKR protocol, matching the $\log |C|$ overhead in the complexity of the proof size. However, the proof size of our second scheme is very close to the GKR protocol. It is only 1.1-1.3× larger, showing that this variant reduces the proof size significantly upon scheme 1. In fact, this overhead is introduced by the second sumcheck protocol to combine multiple points. The term d^2 in the complexity has minimal impact on the total proof size as the constant factor of it is at most 1/2. In all cases, the proof size is succinct. The largest proof size is still less than 1MB and the proof size is always much smaller than the size of the circuit.

As the circuits are generated randomly, the verifier time in all schemes are linear in the circuit size. Therefore, the comparisons on the verifier time of the three protocols are similar to the comparisons on the prover time. As shown in Table 1, our scheme 1 is faster by 4-6× than the original GKR protocol on circuits with $d = 50$, and 5-7× faster for $d = 75$. Our scheme 2 is 17-19× faster on circuits with $d = 50$, and 23-25× faster for $d = 75$. Therefore, we observe in the experiments that our scheme 2 improves the performance of scheme 1 on all the aspects on random circuits, proving our statement in Section 3.3. Compared to the original GKR protocol, our scheme 2 is much faster on the prover time and the verifier time, and incurs a small overhead on the proof size.

5.2 Evaluations of Our Zero Knowledge Argument

In this section, we present the performance of our new zero knowledge argument, Virgo++, for general arithmetic circuits based on the generalized GKR protocol, as described in Section 4. We use the zero knowledge polynomial commitment scheme in [55] to lift our new interactive proofs to zero knowledge arguments. We import the open-source code of zero knowledge polynomial commitment scheme in [9]. We also compare the performance with existing IP and IOP-based zero knowledge proof systems, including Virgo [55], Spartan [42], Aurora [16] and STARK [14].

		Prover time (s)			Verifier time (s)			Proof size (KB)		
		2^9	2^{11}	2^{13}	2^9	2^{11}	2^{13}	2^9	2^{11}	2^{13}
$d = 50$	GKR	0.118	0.465	1.908	0.052	0.206	0.838	83	95	107
	Our Scheme 1	0.043	0.154	0.576	0.013	0.042	0.151	280	397	535
	Our Scheme 2	0.012	0.049	0.197	0.003	0.011	0.044	93	106	120
$d = 75$	GKR	0.244	0.973	3.954	0.100	0.404	1.608	129	147	166
	Our Scheme 1	0.069	0.243	0.910	0.021	0.066	0.237	416	593	803
	Our Scheme 2	0.019	0.075	0.304	0.004	0.017	0.066	168	188	208

Table 1: Comparison of our scheme 1, our scheme 2 and the original GKR on random circuits.

Benchmark. Our benchmark is proving the inference of a neural network for image classification. The zero knowledge proof guarantees that the inference is correctly computed by a committed neural network, while preserving the privacy of the machine learning model. The zero knowledge inference can be used to ensure the validity of machine-learning-as-a-service (MLaaS) on untrusted cloud servers and the reproducibility of the machine learning model, as described in [28, 37, 54]. We use the convolutional neural network of VGGNet [45] on an image of size $224 \times 224 \times 3$ in the ImageNet dataset [27]. The original VGG16 consists of 13 convolutional layers and 3 fully-connected layers. Each convolutional layer is followed by an activation layer and some activation layers are followed by pooling layers.

In our benchmark, we remove all 3 fully-connected layers, as none of Spartan, Aurora and STARK can scale to the large circuits of matrix multiplications in VGG16. Only Virgo and Virgo++ can support them thanks to the efficient sumcheck protocol for matrix multiplications in [46], and we exclude these layers for fair comparison. We further reduce the dimensions of the kernels in the convolutional layers such that Spartan and STARK can run the experiment of VGG16 on our machine. We use the square activation function and the average pooling as in [30, 32], and the structure of the neural network is shown in Table 2. The general arithmetic circuit for the inference of this neural network consists of around 2^{25} gates and 2^{18} inputs. As Virgo and Aurora still cannot scale to this neural network, we further vary the number of convolutional layers from 1 to 13 by selecting subsets of the layers.

Existing systems. We obtain the performance of the existing ZKP schemes as follows:

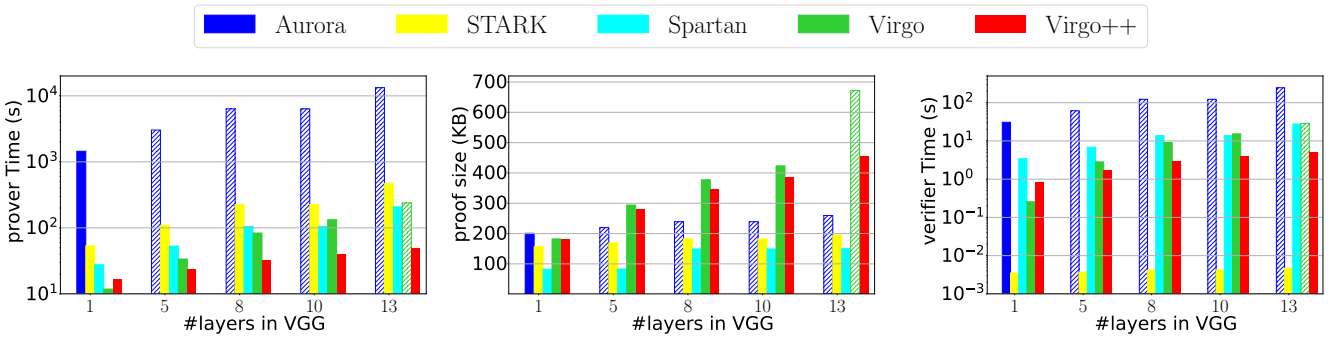
- Virgo: we pad the general arithmetic circuits to layered arithmetic circuits and run the open-source code of Virgo at [9] on these layered circuits. Virgo uses the same field as ours with 100+ bits security.
- Spartan: we convert the general arithmetic circuits to R1CS constraints and run the open-source implementation of Spartan at [7] on the corresponding R1CS. We use the NIZK version of the system, which is in the same setting as all other schemes. Spartan operates on Curve 25519 with 128-bit of security.
- Aurora: we convert the general arithmetic circuits to R1CS constraints and run the open-source implementation of Aurora at [6] on the corresponding R1CS. We use the prime field of 181 bits in Aurora with 128-bit of security.
- STARK: we use the open-source library of Winterfell from Novi Financial [8]. As STARK does not support arithmetic circuit or R1CS, we estimate the performance by running the sample functions in [8] with the same number of multiplications as in the general circuit and R1CS (Fibonacci sequence in the mulfib2/ repo). The field size is 256 bits with 128 bits of security.

Figure 3 shows the prover time, verifier time and the proof size of all schemes. Aurora can only scale to the smallest instance with 1 convolutional layer in the neural network (around 2^{22} R1CS constraints) because of the memory usage. Virgo runs out of the memory for the largest instance because of the large size of the padded circuit. We estimated their performance for larger instances based on the numbers of smaller instances, and the estimations are denoted by the shaded bars in the figure.

Prover time. As shown in in Figure 3(a), Virgo++ (red bars) has the fastest prover time among all of these systems on almost all instances. For the largest instance of VGGNet with 13 convolutional layers (2^{25} gates), the prover of Virgo++ costs only 49.7 seconds. Our prover time is 4.2× faster than Spartan, 9.4× faster than STARK, and 268× faster than Aurora. This is because the circuit is significantly larger than the input (witness), and our generalized GKR protocol on the circuit is very efficient in practice. Our protocol only runs the polynomial commitment scheme on the witness and the input, while Spartan, Aurora and STARK run a similar protocol on the extended witness of the same size as the circuit/R1CS. Our prover time is also 4.8× faster than Virgo, as the corresponding layered circuit after padding is 20× larger (the speedup is 4.8× as both scheme share the same time for the polynomial commitments). In fact, Virgo++ is only slower than Virgo on the smallest instance of 1 convolutional layer, as the layered circuit is almost the same

Input	$224 \times 224 \times 3$
Convolutional	2 kernels of $3 \times 3 \times 3$
Convolutional	2 kernels of $3 \times 3 \times 2$
Pooling	Average pooling of 2×2
Convolutional	4 kernels of $3 \times 3 \times 2$
Convolutional	4 kernels of $3 \times 3 \times 4$
Pooling	Average pooling of 2×2
Convolutional	8 kernels of $3 \times 3 \times 4$
Convolutional	8 kernels of $3 \times 3 \times 8$
Convolutional	8 kernels of $3 \times 3 \times 8$
Pooling	Average pooling of 2×2
Convolutional	16 kernels of $3 \times 3 \times 8$
Convolutional	16 kernels of $3 \times 3 \times 16$
Convolutional	16 kernels of $3 \times 3 \times 16$
Pooling	Average pooling of 2×2
Convolutional	16 kernels of $3 \times 3 \times 16$
Convolutional	16 kernels of $3 \times 3 \times 16$
Convolutional	16 kernels of $3 \times 3 \times 16$
Pooling	Average pooling of 2×2

Table 2: The structure of the VGGNet in our experiments.



(a) $\mathcal{P}$ time (b) Proof size (c) $\mathcal{V}$ time

Figure 3: Comparison of Virgo++ and existing IOP-based ZKP systems. Shaded bars denote estimations.

as the general circuit. The speedup will become larger on neural networks with larger dimensions and depth, however, none of the existing schemes can further scale up.

Proof size. Figure 3(b) shows the proof size of the schemes. As shown in the figure, the proof size of Virgo++ is relatively large among all the schemes. Our proof size is always the same as or smaller than Virgo, as they have the same number of layers and the same polynomial commitments, while the layered circuit in Virgo is larger. The result justifies that the overhead of our generalized GKR protocol over the original GKR on the proof size is very small in practice. Comparing to other schemes, our proof size is 0.9-1.7× of Aurora, 1.2-2.3× larger than STARK, 2.2-3× larger than Spartan. However, the proof size of all schemes are more than 100KB in almost all instances. This is because the proof size of the IP-based and IOP-based ZKP schemes is large in general. Therefore, we believe it is a reasonable trade-off for better prover time in our scheme in practice.

The proof size of Spartan is 82KB in the VGG with 1 layer as it uses the polynomial commitment based on discrete-log in [50]. However, the proof size soon becomes more than 100KB and will be larger than ours on a larger neural network as it is square-root in the size of the circuit. Moreover, our scheme as well as other IOP-based schemes are post-quantum secure.

Verifier time. Figure 3(c) shows the verifier time. Other than STARK, the comparison of the verifier time in all schemes is similar to that of the prover time. This is because the verifier time of Virgo++, Virgo, Spartan, Aurora are all linear in the size of the circuit/R1CS. For the VGGNet of 13 convolutional layers, the verifier of Virgo++ 2.7× faster than Spartan, 5.7× faster than Virgo and 46.8× faster than Aurora. As the circuit/R1CS for VGG inference is well-structured, we believe all of these system can achieve sublinear verifier time on this computation. However, the sublinear verifier for structured computations is not considered in the existing implementations, and is left as a future work.

The verifier time in STARK is significantly faster than others and is only milliseconds in the experiments. This is because it represents computations in an algebraic intermediate representation similar to a random access machine with uniformity. However, as the performance of STARK is estimated using a much simpler computation than neural network inferences, the verifier time is not comparable in this experiment.

Discussion. As demonstrated in the experiments, our new ZKP scheme has efficient prover time for computations where the circuit size is much larger than the size of the input and the witness. Common computations and applications in practice include matrix multiplications, convolutions, neural network inferences and training, image processing and transformations [39] and RAM-to-circuit reductions in RAM-based ZKP schemes [15, 17, 20, 58].

Comparing to ZKP schemes based on the original GKR protocol, the performance of our new scheme is similar or better on almost all computations. Therefore, our new scheme is useful on the design of the frontend compiling the statement of zero knowledge proofs to an arithmetic circuit. Prior ZKP schemes based on GKR protocols only work for layered circuits and it is very inconvenient to turn computations to strictly layered circuits efficiently. All existing implementations such as [25, 46, 50, 53, 58] create their own data structures to represent layered circuits with function pointers denoting the wiring predicates. In contrast, our scheme works for general circuits and our system can directly take circuits in commonly-used formats such as the Bristol format [4]. With this feature, we can build on existing frontends in the literature such as the EMP toolkit [52] to compile the statement to our zero knowledge proof, and also take the advantage of existing highly optimized circuits. Other frontends such as Jsnark [2] and Buffet [1] for R1CS can also be modified to output general arithmetic circuits.

Similar to all IP-based and IOP-based ZKP schemes, the proof size of our scheme is relatively large. Thus the scheme should be used in applications where the proof size is not the bottleneck to improve the prover efficiency, such as blockchains with zero knowledge proofs in the offline storage [3], and zero knowledge program analysis and vulnerability disclosures [5, 41].

ACKNOWLEDGMENT

We greatly thank Yuval Ishai for proposing the interesting problem of interactive proofs for general circuits and for the helpful discussions on the paper. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. TWC-1518899, DARPA under Grant No. N66001-15-C-4066 and DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA.

REFERENCES

[1] 2014. Buffet. https://github.com/pepper-project/releases. (2014).
[2] 2015. jsnark. https://github.com/akosba/jsnark. (2015).
[3] 2017. Oasis Labs. https://www.oasislabs.com/. (2017).
[4] 2018. Bristol Circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/. (2018).
[5] 2019. DARPA SIEVE program. https://www.darpa.mil/news-events/2019-07-18. (2019).
[6] 2020. libiop. https://github.com/scipr-lab/libiop. (2020).
[7] 2020. Spartan. https://github.com/microsoft/Spartan. (2020).
[8] 2020. STARK implementation. https://github.com/novifinancial/winterfell. (2020).
[9] 2020. Virgo implementation. https://github.com/sunblaze-ucb/Virgo. (2020).
[10] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.
[11] László Babai, Lance Fortnow, and Carsten Lund. 1991. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity* 1, 1 (1991), 3–40.
[12] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël Del Pino, Jens Groth, and Vadim Lyubashevsky. 2018. Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits. In *Annual International Cryptology Conference*. Springer, 669–699.
[13] Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 263–280.
[14] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*. Springer, 701–732.
[15] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*.
[16] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. 2019. Aurora: Transparent succinct arguments for R1CS. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 103–128.
[17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *Proceedings of the USENIX Security Symposium, 2014*.
[18] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. 2014. Verifiable computation using multiple provers. Cryptology ePrint Archive, Report 2014/846. (2014). https://eprint.iacr.org/2014/846.
[19] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *International Conference on the Theory and Applications of Cryptographic Techniques*.
[20] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. 2018. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 595–626.
[21] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *Proceedings of the Symposium on Security and Privacy (SP), 2018*, Vol. 00. 319–338.
[22] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1825–1842.
[23] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. 2017. A Zero Knowledge Sumcheck and its Applications. *CoRR* abs/1704.02086 (2017). arXiv:1704.02086 http://arxiv.org/abs/1704.02086
[24] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical Verified Computation with Streaming Interactive Proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*.
[25] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*. 90–112.
[26] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile Verifiable Computation. In *S&P 2015*.
[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[28] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: Efficient Zero-Knowledge Proofs for Neural Networks. Cryptology ePrint Archive, Report 2021/087. (2021). https://eprint.iacr.org/2021/087.
[29] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. 2016. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.
[30] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*. 4672–4681.
[31] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits.. In *USENIX Security Symposium*. 1069–1083.
[32] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.
[33] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4, Article 27 (Sept. 2015), 64 pages.
[34] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.
[35] Jens Groth. 2009. Linear algebra with sub-linear zero-knowledge arguments. In *Advances in Cryptology-CRYPTO 2009*. Springer, 192–208.
[36] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *Proceedings of the annual ACM symposium on Theory of computing*. ACM, 21–30.
[37] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs. Cryptology ePrint Archive, Report 2020/584. (2020). https://eprint.iacr.org/2020/584.
[38] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (Oct. 1992), 859–868.
[39] Assa Naveh and Eran Tromer. 2016. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 255–271.
[40] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *S&P 2013*. 238–252.
[41] Ben Perez. 2020. Reinventing Vulnerability Disclosure using Zero-knowledge Proofs. https://securityboulevard.com/2020/05/reinventing-vulnerability-disclosure-using-zero-knowledge-proofs/. (2020).
[42] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.
[43] Srinath Setty and Jonathan Lee. 2020. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275. (2020). https://eprint.iacr.org/2020/1275.
[44] Adi Shamir. 1992. Ip= pspace. *Journal of the ACM (JACM)* 39, 4 (1992), 869–877.
[45] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[46] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.).
[47] Riad S Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. 2016. Verifiable asics. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 759–778.
[48] Riad S Wahby, Ye Ji, Andrew J Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. 2017. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM.
[49] Riad S Wahby, Srinath TV Setty, Zuocheng Ren, Andrew J Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation.. In *NDSS*.
[50] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 926–943.
[51] Michael Walfish and Andrew J. Blumberg. 2015. Verifying computations without reexecuting them. *Commun. ACM* 58, 2 (2015), 74–84.
[52] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2020. EMP-toolkit: Efficient Multiparty computation toolkit. https://github.com/emp-toolkit. (2020).
[53] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Annual International Cryptology Conference*. Springer, 733–764.
[54] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero Knowledge Proofs for Decision Tree Predictions and Accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2039–2053.
[55] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *S&P 2020*.
[56] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 863–880.
[57] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology

ePrint. (2017).
[58] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. vRAM: Faster verifiable RAM with program-independent preprocessing. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)*.

A THE GKR PROTOCOL ON LAYERED CIRCUITS

The formal protocol is given in Protocol 2.

B THE PROTOCOL TO COMBINE MULTIPLE CLAIMS BY AN ARITHMETIC CIRCUIT.

The formal protocol is given in Protocol 3.

Efficiency. In order to analyze the prover time for the protocol (Protocol 3), we consider the specific structure of C_i, as shown in Figure 2. For layer $k < i$, there are $S_{k,i}$ gates from layer i connected to layer k. The number of gates in C_i is at most $8|\mathbf{V}|$, which is $8\sum_{k=0}^{i-1} S_{k,i}$. By Theorem 2.4, the prover time for the circuit C_i is $O(\sum_{k=0}^{i-1} S_{k,i})$. So the total prover time for circuits $C_1, \ldots, C_d$ is $8\sum_{i=1}^{d}\sum_{k=0}^{i-1} S_{k,i} = O(|C|)$ as $\sum_{i=1}^{d}\sum_{k=0}^{i-1} S_{k,i}$ equals to the number of all output wires in the circuit, which is at most $2|C|$.

The size of C_i is $O(S_{0,i} + \ldots + S_{i-1,i}) = O(|C|)$, the depth of C_i is $O(\log S_i) = O(\log |C|)$ and the size of input $\mathbf{R}_i$ is at most $s_{0,i} + s_{1,i} + \ldots + s_{i-1,i} + s_{i-1,i} \leq d \log |C|$. Let Q_i be the time to evaluate all $\tilde{add}$ and $\tilde{mult}$ at the corresponding random points in C_i. Therefore, the verifier time for C_i is $O(\log^2 |C| + d \log |C| + Q_i)$ and the proof size is $O(\log^2 |C|)$ by Theorem 2.4. In total for all layers, $\mathcal{V}$ runs in $\min\{O(d \log^2 |C| + d^2 \log |C| + Q), |C|\}$ time and the proof size is $\min\{O(d \log^2 |C|), |C|\}$, where $Q = Q_1 + Q_2 + \ldots + Q_d$.

C THE FULL PROTOCOL FOR GENERAL ARITHMETIC CIRCUITS AND THE PROOF

Our full protocol is given in Protocol 4.

Proof of Theorem 3.8. **Completeness**. The completeness is straightforward by the completeness of the sumcheck protocol.

Soundness. For the soundness, for any PPT adversary $\mathcal{A}$, we use $\tilde{V}'$ to represent the correct messages corresponding to $\tilde{V}$ in Protocol 4 with input **in** and the correct execution for circuit C. Suppose $C(\mathbf{in}) \neq \mathbf{out}$, there must exist a layer i such that $\tilde{V}_j(r^{(j)}) = \tilde{V}'_j(r^{(j)})$ and $\tilde{V}_{k,j}(r^{(k,j)}) = \tilde{V}'_{k,j}(r^{(k,j)})$ for $j > i$ and all $k < j$ but $\tilde{V}_i(r^{(i)}) \neq \tilde{V}'_i(r^{(i)})$ or $\tilde{V}_{k,i}(r^{(k,i)}) \neq \tilde{V}'_{k,i}(r^{(k,i)})$ for some $k < i$, which event is defined as E_i. This event can be divided into three cases:

- Case 1: The random elements are chosen in a way such that $\sum_{k=0}^{i-1} \alpha_{k,i}\tilde{V}'_{k,i}(r^{(k,i)}) + \alpha'_{i-1,i}\tilde{V}'_{i-1,i}(r^{(i-1,i)'}) = \sum_{k=0}^{i-1} \alpha_{k,i}\tilde{V}_{k,i}(r^{(k,i)}) + \alpha'_{i-1,i}\tilde{V}_{i-1,i}(r^{(i-1,i)'})$. This happens with probability at most $\frac{1}{|\mathbb{F}|}$.
- Case 2: The above case does not happen, but at the end of sumcheck protocol induced by Equation 9, the final round random evaluation (e.g. $\tilde{V}_i(r^{(i)})$) is consistent with the single evaluation of $\tilde{V}'_i(r^{(i)})$. By the soundness of sumcheck protocol this happens with probability at most $\frac{2\lceil \log S_i \rceil}{|\mathbb{F}|}$.
- Case 3: We have $\tilde{V}_i(r^{(i)}) \neq \tilde{V}'_i(r^{(i)})$, but the verifier accepts after the sumcheck protocol for $\tilde{V}_i(r^{(i)})$. Since we assume that $\tilde{V}_j(r^{(j)}) = \tilde{V}'_j(r^{(j)})$ and $\tilde{V}_{k,j}(r^{(k,j)}) = \tilde{V}'_{k,j}(r^{(k,j)})$ for $j > i$ and all $k < j$, this happens with probability at most $\frac{2\lceil \log S_{i+1} \rceil}{|\mathbb{F}|}$ by the soundness of sumcheck protocol.

Thus the overall probability that event E_i happens is at most $\frac{2\lceil \log S_i \rceil}{|\mathbb{F}|} + \frac{2\lceil \log S_{i+1} \rceil}{|\mathbb{F}|} + \frac{1}{|\mathbb{F}|} = O(\frac{\log |C|}{|\mathbb{F}|})$.

Eventually, by the union bound, we have the following statement:

$$\begin{aligned}
&\Pr[C(\mathbf{in}) \neq \mathbf{out} \wedge \mathcal{V} \text{ outputs } 1] \\
\leq &\Pr[\exists i, E_i] \\
\leq &\Pr[E_0] + \Pr[E_1] + \ldots + \Pr[E_{d-1}] \\
\leq &O(\frac{\log |C|}{|\mathbb{F}|}) + O(\frac{\log |C|}{|\mathbb{F}|}) + \ldots + O(\frac{\log |C|}{|\mathbb{F}|}) \\
\leq &O(\frac{d \log |C|}{|\mathbb{F}|})
\end{aligned}$$

The efficiency follows the efficiency analysis of Section 3.2 and Section 3.3.

D OUR NEW ZERO KNOWLEDGE ARGUMENT SCHEME

D.1 Definitions

We introduce definitions of zero knowledge arguments and zero knowledge polynomial commitments before presenting the formal protocols.

Zero knowledge arguments. An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in R$ for some input x. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w. We use $\mathcal{G}$ to represent the generation phase of the public parameters pp. Formally, consider the definition below, where we assume R is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition D.1. Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero knowledge argument of knowledge for $\mathcal{R}$ if the following holds.

- **Correctness**. For every pp output by $\mathcal{G}(1^\lambda)$ and $(x, w) \in R$,
$$\langle \mathcal{P}(\text{pp}, w), \mathcal{V}(\text{pp}) \rangle(x) = 1$$
- **Knowledge Soundness**. For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that given the access to the entire executing process and the randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that $\text{pp} \leftarrow \mathcal{G}(1^\lambda)$, $\pi^* \leftarrow \mathcal{P}^*(x, \text{pp})$ and $w \leftarrow \mathcal{E}||\mathcal{P}^*(\text{pp}, x, \pi^*)$, the following probability is $\text{negl}(\lambda)$:
$$\Pr[(x; w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \text{pp}) = 1]$$
- **Zero knowledge**. There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0,1\}^*$, $(x; w) \in \mathcal{R}$, pp output by $\mathcal{G}(1^\lambda)$, it holds that
$$\text{View}(\langle \mathcal{P}(\text{pp}, w), \mathcal{V}^*(z, \text{pp}) \rangle(x)) \approx \mathcal{S}^{\mathcal{V}^*}(x, z)$$

We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a **succinct** argument system if the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) are $\text{poly}(\lambda, |x|, \log |w|)$.

Protocol 2 (GKR). *Let $\mathbb{F}$ be a finite field. Let $C: \mathbb{F}^n \to \mathbb{F}^k$ be a d-depth layered arithmetic circuit. $\mathcal{P}$ wants to convince that $\boldsymbol{out} = C(\boldsymbol{in})$ where $\boldsymbol{in}$ is the input from $\mathcal{V}$, and $\boldsymbol{out}$ is the output. Without loss of generality, assume n and k are both powers of 2 and we can pad them if not.*

(1) Define the multilinear extension of array $\boldsymbol{out}$ as $\tilde{V}_0$. $\mathcal{V}$ chooses a random $g \in \mathbb{F}^{s_0}$ and sends it to $\mathcal{P}$. Both parties compute $\tilde{V}_0(g)$.

(2) $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on

$$\tilde{V}_0(g^{(0)}) = \sum_{x,y \in \{0,1\}^{s_1}} (\tilde{add}_1(g^{(0)}, x, y)(\tilde{V}_1(x) + \tilde{V}_1(y)) + \tilde{mult}_1(g^{(0)}, x, y)\tilde{V}_1(x)\tilde{V}_1(y))$$

At the end of the protocol, $\mathcal{V}$ receives $\tilde{V}_1(u^{(1)})$ and $\tilde{V}_1(v^{(1)})$. $\mathcal{V}$ computes $\tilde{mult}_1(g^{(0)}, u^{(1)}, v^{(1)})$, $\tilde{add}_1(g^{(0)}, u^{(1)}, v^{(1)})$ and checks that $\tilde{add}_1(g^{(0)}, u^{(1)}, v^{(1)})\,(\tilde{V}_1(u^{(1)}) + \tilde{V}_1(v^{(1)})) + \tilde{mult}_1(g^{(0)}, u^{(1)}, v^{(1)})\,\tilde{V}_1(u^{(1)})\tilde{V}_1(v^{(1)})$ equals to the last message of the sumcheck.

(3) For $i = 1, \ldots, d-1$:

- *$\mathcal{V}$ randomly selects $\alpha_{i,1}, \alpha_{i,2} \in \mathbb{F}$ and sends them to $\mathcal{P}$.*
- *$\mathcal{P}$ and $\mathcal{V}$ run the sumcheck on the equation*

$$\alpha_{i,1}\tilde{V}_i(u^{(i)}) + \alpha_{i,2}\tilde{V}_i(v^{(i)}) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} ((\alpha_{i,1}\tilde{add}_{i+1}(u^{(i)}, x, y) + \alpha_{i,2}\tilde{add}_{i+1}(v^{(i)}, x, y))(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y))$$
$$+(\alpha_{i,1}\tilde{mult}_{i+1}(u^{(i)}, x, y) + \alpha_{i,2}\tilde{mult}_{i+1}(v^{(i)}, x, y))\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))$$

- *At the end of the sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ $\tilde{V}_{i+1}(u^{(i+1)})$ and $\tilde{V}_{i+1}(v^{(i+1)})$.*
- *$\mathcal{V}$ computes the following and checks if it equals to the last message of the sumcheck. For simplicity, let $Mult_{i+1}(x) = \tilde{mult}_{i+1}(x, u^{(i+1)}, v^{(i+1)})$ and $Add_{i+1}(x) = \tilde{add}_{i+1}(x, u^{(i+1)}, v^{(i+1)})$.*

$$(\alpha_{i,1}Mult_{i+1}(u^{(i)}) + \alpha_{i,2}Mult_{i+1}(v^{(i)})(\tilde{V}_{i+1}(u^{(i+1)})\tilde{V}_{i+1}(v^{(i+1)}))+$$
$$(\alpha_{i,1}Add_{i+1}(u^{(i)}) + \alpha_{i,2}Add_{i+1}(v^{(i)})(\tilde{V}_{i+1}(u^{(i+1)}) + \tilde{V}_{i+1}(v^{(i+1)}))$$

If all checks in the sumcheck pass, V uses $\tilde{V}_{i+1}(u^{(i+1)})$ and $\tilde{V}_{i+1}(v^{(i+1)})$ to proceed to the $(i+1)$-th layer. Otherwise, $\mathcal{V}$ outputs $\mathtt{0}$ and aborts.

(4) At the input layer d, $\mathcal{V}$ has two claims $\tilde{V}_d(u^{(d)})$ and $\tilde{V}_d(v^{(d)})$. $\mathcal{V}$ evaluates $\tilde{V}_d$ at $u^{(d)}$ and $v^{(d)}$ using the input and checks that they are the same as the two claims. If yes, output $\mathtt{1}$; otherwise, output $\mathtt{0}$.

Figure 4: The GKR protocol.

Protocol 3. *Let C_i be the circuit in Figure 2 with input $\boldsymbol{in}$ consisting of two parts: $\boldsymbol{V}_i = (V_i(0), \ldots, V_i(S_i - 1))$ and $\boldsymbol{R} = (r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'})$, and the output $\boldsymbol{out} = (\tilde{V}_{0,i}(r^{(0,i)}), \ldots, \tilde{V}_{i-1,i}(r^{(i-1,i)}), \tilde{V}_{i-1,i}(r^{(i-1,i)'}))$. We use $\boldsymbol{V} = (\boldsymbol{V}_{0,i}, \ldots, \boldsymbol{V}_{i-1,i}, \boldsymbol{V}_{i-1,i})$ to represent subsets of $\boldsymbol{V}_i$ used in layer j $(j < i)$, and $\boldsymbol{T}_R = (T_{r^{(0,i)}}, \ldots, T_{r^{(i-1,i)}}, T_{r^{(i-1,i)'}})$ to represent bookkeeping tables after expanding $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$.*

- *$\mathcal{P}$ and $\mathcal{V}$ invoke Protocol 2 on inner products to reduce the claim about $\boldsymbol{out}$ to the claim about the layer of $\boldsymbol{V}$ and $\boldsymbol{T}_R$: $q = r_1 \cdot \boldsymbol{V}(r) + (1 - r_1) \cdot \boldsymbol{T}_R(r)$*
- *$\mathcal{V}$ requires $\mathcal{P}$ to provide values of $\boldsymbol{V}(r)$ and $\boldsymbol{T}_R(r)$ to check $q = r_1 \cdot \boldsymbol{V}(r) + (1 - r_1) \cdot \boldsymbol{T}_R(r)$.*
- *$\mathcal{P}$ and $\mathcal{V}$ invoke Protocol 2 on the left part and the right part of C_i as shown in Figure 2, separately. For the left part, it reduces the claim about $\boldsymbol{V}(r)$ to the claim about $\boldsymbol{V}_i(r^{(i)})$ in one layer. For the right part, it reduces the claim about $\boldsymbol{T}_R(r)$ to the claim about $\boldsymbol{R}(r^{(i)})$.*
- *$\mathcal{V}$ asks $\mathcal{P}$ to send $\boldsymbol{V}_i(r^{(i)})$ and checks the reduction for the left part. $\mathcal{V}$ computes $\boldsymbol{R}(r^{(i)})$ itself and checks the reduction for the right part. If both checks pass, output $\mathtt{1}$; otherwise, output $\mathtt{0}$.*

In the definition of zero knowledge, $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ is given the randomness of $\mathcal{V}^*$ sampled from polynomial-size space. This definition is commonly used in existing transparent zero knowledge proof schemes [10, 16, 21, 50, 53, 55].

Zero knowledge polynomial commitment. Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$, and D be a variable-degree parameter. We use $\mathcal{W}_{\ell,d}$ to denote the collection of all monomials in $\mathcal{F}$ and $N = |\mathcal{W}_{\ell,D}| = (D+1)^\ell$. A zero knowledge verifiable polynomial commitment (zkPC) for $f \in \mathcal{F}$ and $t \in \mathbb{F}^\ell$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$,
- $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$,
- $((y, \pi); \{\mathtt{0}, \mathtt{1}\}) \leftarrow \langle \mathsf{zkPC.Open}(f, r_f), \mathsf{zkPC.Verify}(\mathsf{com}) \rangle (t, \mathsf{pp})$

Definition D.2. A zkPC scheme satisfies the following properties:

- **Completeness.** For any polynomial $f \in \mathcal{F}$ and value $t \in \mathbb{F}^\ell$, $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$, it holds that

$$\Pr\left[\langle \mathsf{zkPC.Open}(f, r_f), \mathsf{zkPC.Verify}(\mathsf{com}) \rangle (t, \mathsf{pp}) = 1\right] = 1$$

- **Knowledge Soundness.** For any PPT adversary $\mathcal{A}$, $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, there exists a PPT extractor $\mathcal{E}$. Given any tuple (pp, com^*) and the executing process of $\mathcal{A}$, $\mathcal{E}$ can extract a function $f^* \in \mathcal{F}$ and the randomness r_{f^*} such that $(f^*, r_{f^*}) \leftarrow \mathcal{E}||\mathcal{A}(\mathsf{pp}, \mathsf{com}^*)$ and $\mathsf{com}^* \leftarrow \mathsf{zkPC.Commit}(f^*, r_{f^*}, \mathsf{pp})$. The

Protocol 4 (Generalized GKR). *Let $\mathbb{F}$ be a prime field. Let $C: \mathbb{F}^n \to \mathbb{F}^k$ be a d-depth general arithmetic circuit. $\mathcal{P}$ wants to convince $\mathcal{V}$ that $C(\boldsymbol{in}) = \boldsymbol{out}$ where $\boldsymbol{in}$ is the input and $\boldsymbol{out}$ is the output. Without loss of generality, assume n is a power of 2 and both parties can pad them if not.*

(1) Define the multilinear extension of array $\boldsymbol{out}$ as $\tilde{V}_0$. $\mathcal{V}$ chooses a random $g \in \mathbb{F}^{s_0}$ and sends it to $\mathcal{P}$. Both parties compute $\tilde{V}_0(g)$.

(2) $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on Equation 6 for $i = 0$. At the end of the protocol, $\mathcal{V}$ receives $\tilde{V}_{0,1}(r^{(0,1)'})$, $\tilde{V}_{0,1}(r^{(0,1)})$, $\tilde{V}_{0,2}(r^{(0,2)})$, $\ldots$, $\tilde{V}_{0,d}(r^{(0,d)})$. $\mathcal{V}$ computes left side of the above equation by removing the summation symbol and replacing x, y with $r^{(0,1)'}, r^{(0,1)}$. If it does not equal to the last message of the sumcheck, $\mathcal{V}$ outputs 0 and aborts.

(3) For $i = 1, \ldots, d-1, d$:

(a) Given $\tilde{V}_{0,i}(r^{(0,i)}), \ldots, \tilde{V}_{i-1,i}(r^{(i-1,i)'}), V_{i-1,i}(r^{(i-1,i)})$ and $r^{(0,i)}, r^{(1,i)}, \ldots, r^{(i-1,i)'}, r^{(i-1,i)}$, $\mathcal{V}$ chooses $i+1$ random elements $\alpha_{0,i}, \ldots, \alpha_{i-1,i}, \alpha'_{i-1,i}$ in $\mathbb{F}$ and sends them to $\mathcal{P}$. Then $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck protocol on Equation 9. $\mathcal{V}$ receives $\tilde{V}_i(r^{(i)})$ for some randomness $r^{(i)} \in \mathbb{F}^{s_i}$ in the last round if he does not abort.

(b) If $i < d$, $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck on Equation 6 by replacing g with $r^{(i)}$. At the end of the sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ $\tilde{V}_{i,i+1}(r^{(i,i+1)'})$, $\tilde{V}_{i,i+1}(r^{(i,i+1)})$, $\ldots$, $\tilde{V}_{i,d}(r^{(i,d)})$.
$\mathcal{V}$ computes the left side of the above equation by removing the summation symbol and replacing x, y with $r^{(i,i+1)'}, r^{(i,i+1)}$ and checks it equals to the last message of the sumcheck. If all checks in the sumcheck pass, V uses $\tilde{V}_{0,i+1}(r^{(0,i+1)}), \ldots, \tilde{V}_{i,i+1}(r^{(i,i+1)'})$ and $\tilde{V}_{i,i+1}(r^{(i,i+1)})$ to proceed to the $(i+1)$-th layer. Otherwise, $\mathcal{V}$ outputs 0 and aborts.

(4) At the input layer d, $\mathcal{V}$ has one claim of $\tilde{V}_d(r^{(d)})$. $\mathcal{V}$ computes it locally or queries the oracle of evaluations of $\tilde{V}_d$ at $r^{(d)}$ and checks if it is the same as the claim. If yes, output 1; otherwise, output 0.

Figure 5: Our generalized GKR protocol.

following probability is negligible of λ:

$$\Pr[((y^*, \pi^*); 1) \leftarrow \langle \mathcal{A}(), \mathsf{zkPC.Verify}(\mathsf{com}^*)\rangle(t, \mathsf{pp}) \\ \wedge (f^*, r_{f^*}) \leftarrow \mathcal{E}||\mathcal{A}(\mathsf{pp}, \mathsf{com}^*) \wedge f^*(t) \neq y^*]$$

- **Zero Knowledge.** For security parameter λ, polynomial $f \in \mathcal{F}$, $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:

$\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp})$:	$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp})$:
(1) $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$	(1) $\mathsf{com} \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{pp})$
(2) $t \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{pp})$	(2) $t \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{pp})$
(3) $(y, \pi) \leftarrow \langle \mathsf{zkPC.Open}(f, r_f), \mathcal{A}\rangle(t, \mathsf{pp})$	(3) $(y, \pi) \leftarrow \langle \mathcal{S}_2, \mathcal{A}\rangle(t_i, \mathsf{pp})$, given oracle access to $y = f(t)$.
(4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi, \mathsf{pp})$	(4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi, \mathsf{pp})$
(5) Output b	(5) Output b

For any PPT algorithm $\mathcal{A}$ and all polynomial $f \in \mathbb{F}$, there exists simulator $\mathcal{S}$ such that

$$|\Pr[\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp}) = 1]| \leq \mathsf{negl}(\lambda).$$

D.2 Zero Knowledge Sumcheck

To build a zero knowledge argument for arbitrary arithmetic circuit using Protocol 4, we follow the same blueprint of [53] using zkPC, zero knowledge sumcheck and low degree extensions. In the following, we present the zero knowledge version of step 3(b) and step 3(a) in Protocol 4, followed by the whole zero knowledge argument.

In step 3(b) of the full protocol, $\mathcal{P}$ and $\mathcal{V}$ execute a sumcheck protocol on Equation 6, during which $\mathcal{P}$ sends $\mathcal{V}$ evaluations of the polynomial at several random points chosen by $\mathcal{V}$. These evaluations leak information about the values in the circuit, as they can be viewed as weighted sums of these values.

To prevent the leakage, we take the zero knowledge sumcheck proposed by Xie et al. in [53]. To prove

$$H = \sum\nolimits_{x_1, x_2, \ldots, x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell),$$

the prover generates a random polynomial g such that $g(x_1, \ldots, x_\ell) = a_0 + g_1(x_1) + g_2(x_2) + \ldots + g_\ell(x_\ell)$, where $g_i(x_i) = a_{i,1}x_i + a_{i,2}x_i^2 + \ldots + a_{i,\tau}x_i^\tau$ is a random univariate polynomial of degree τ (τ is the variable degree of f). Note here that the size of g is only $O(\tau\ell)$, while the size of f is exponential in ℓ. $\mathcal{P}$ commits to the polynomial g using zkPC.Commit, and sends the verifier a claim $G = \sum_{x_1, x_2, \ldots, x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell)$. The verifier picks a random number $\rho \in \mathbb{F}$, and execute a sumcheck protocol with the prover on $H + \rho G = \sum_{x_1, x_2, \ldots, x_\ell \in \{0,1\}} (f(x_1, x_2, \ldots, x_\ell) + \rho g(x_1, x_2, \ldots, x_\ell))$.

At the last round of this sumcheck, the prover opens the commitment of g at $g(r_1, \ldots, r_\ell)$ using zkPC.Open, and the verifier computes $f(r_1, \ldots, r_\ell)$ by subtracting $\rho g(r_1, \ldots, r_\ell)$ from the last message, and compares it with the oracle access of f. It is shown that as long as the polynomial commitment is zero knowledge, the protocol is zero knowledge. Intuitively, this is because the information of f transmitted in the sumcheck protocol is exactly masked by the randomness of g. We present the protocol in Protocol 5 and we have the following theorem:

Theorem D.3 ([53]). *Protocol 5 is complete and sound for the relationship of $H = \sum_{x_1, x_2, \ldots, x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell)$. In addition, for every verifier $\mathcal{V}^*$ and every ℓ-variate polynomial $f: \mathbb{F}^\ell \to \mathbb{F}$ with variable degree d, there exists a simulator $\mathcal{S}$ such that given access to $H = \sum_{x_1, x_2, \ldots, x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell)$, $\mathcal{S}$ is able to simulate the partial view of $\mathcal{V}^*$ in Protocol 5. The efficiency of prover time, verifier time and proof size in Protocol 5 retain the same as in Protocol 1.*

We apply the zero knowledge sumcheck directly on the sumcheck equation (Equation 6 and 9) of our new GKR protocol. It eliminates all the leakage during the sumcheck protocol.

D.3 Zero Knowledge GKR

Even with the zero knowledge sumcheck, the protocol still leaks information about values in the circuit. In particular, at the end

Protocol 5 (Zero knowledge sumcheck). *We assume the existence of a zkPC protocol defined in Section D.1. For simplicity, we omit the randomness r_f and public parameters* pp, vp *without any ambiguity. To prove the claim* $H = \sum_{x_1,x_2,\ldots,x_\ell\in\{0,1\}} f(x_1,x_2,\ldots,x_\ell)$*:*

(1) $\mathcal{P}$ *selects a polynomial* $g(x_1,\ldots,x_\ell) = a_0 + g_1(x_1) + g_2(x_2) + \ldots + g_\ell(x_\ell)$*, where* $g_i(x_i) = a_{i,1}x_i + a_{i,2}x_i^2 + \ldots + a_{i,\tau}x_i^\tau$ *and all* $a_{i,j}$*s are uniformly random.* $\mathcal{P}$ *sends* $H = \sum_{x_1,x_2,\ldots,x_\ell\in\{0,1\}} f(x_1,x_2,\ldots,x_\ell)$, $G = \sum_{x_1,x_2,\ldots,x_\ell\in\{0,1\}} g(x_1,x_2,\ldots,x_\ell)$ *and* $\mathsf{com}_g = \mathsf{zkPC.Commit}(g, r_g, \mathsf{pp})$ *to* $\mathcal{V}$.

(2) $\mathcal{V}$ *uniformly selects* $\rho \in \mathbb{F}^*$*, computes* $H + \rho G$ *and sends* ρ *to* $\mathcal{P}$.

(3) $\mathcal{P}$ *and* $\mathcal{V}$ *run the sumcheck protocol on*

$$H+\rho G = \sum_{x_1,x_2,\ldots,x_\ell\in\{0,1\}} (f(x_1,x_2,\ldots,x_\ell)+\rho g(x_1,x_2,\ldots,x_\ell))$$

(4) *At the last round of the sumcheck protocol,* $\mathcal{V}$ *obtains a claim* $h_\ell(r_\ell) = f(r_1,r_2,\ldots,r_\ell) + \rho g(r_1,r_2,\ldots,r_\ell)$*.* $\mathcal{P}$ *opens the commitment of* g *at* $r = (r_1,\ldots,r_\ell)$ *and* $\mathcal{V}$ *verifies by using* zkPC.Open *and* zkPC.Verify*. If the verification fails,* $\mathcal{V}$ *aborts.*

(5) $\mathcal{V}$ *computes* $h_\ell(r_\ell) - \rho g(r_1,\ldots,r_\ell)$ *and compares it with the oracle access of* $f(r_1,\ldots,r_\ell)$.

Figure 6: Zero knowledge sumcheck protocol.

of the zero knowledge sumcheck, $\mathcal{V}$ still needs an oracle access to $f(r_1,\ldots,r_\ell)$. When executed on Equation 6, the verifier evaluates all $\tilde{add}$ and $\tilde{mult}$ at the random point, and queries the prover for the evaluations of $\tilde{V}_{i,i+1},\cdots,\tilde{V}_{i,d}$. These evaluations reveal information about values in the circuit.

To prevent this leakage, we use the same idea in [53] to replace them with their low-degree extensions $\dot{V}_{i,i+1},\cdots,\dot{V}_{i,d}$. Let

$$\dot{V}_{i,j}(x) \overset{def}{=} \tilde{V}_{i,j}(x) + Z_{i,j}(x)\cdot\sum_{w\in\{0,1\}} R_{i,j}(x_1,w), \tag{10}$$

where $Z_{i,j}(x) = \prod_{k=1}^{s_{i,j}} x_k(1-x_k)$ is the vanishing polynomial, i.e., $Z_{i,j}(x) = 0$ for all $x \in \{0,1\}^{s_{i,j}}$, and $R_{i,j}$ is the mask polynomial with only two variables generated by $\mathcal{P}$.

Additionally, in the last round of the sumcheck on Equation 9, $\mathcal{V}$ asks for $\tilde{V}_i(r^{(i)})$, which leaks information about V_i. With exactly the same idea as above, we replace it with its low-degree extension $\dot{V}_i$ such that

$$\dot{V}_i(x_1,\ldots,x_{s_i}) \overset{def}{=} \tilde{V}_i(x_1,\ldots,x_{s_i}) + Z_i(x_1,\ldots,x_{s_i})\sum_{w\in\{0,1\}} R_i(x_1,w),$$

where $Z_i(x) = \prod_{i=1}^{s_i} x_i(1-x_i)$ is still the vanishing polynomial, and R_i is still a mask multilinear polynomial with only two variables. As $R_{i,i+1},\cdots,R_{i,d}$ and R_i are randomly selected by $\mathcal{P}$, revealing several evaluations of them does not leak information about $V_{i,i+1},\cdots,V_{i,d}$ and V_i thus the values in the circuit. The zero knowledge polynomial commitment scheme is used to commit to these masking polynomials and later open them at random points. With these changes, Equation 6 becomes

$$\begin{aligned}\dot{V}_i(g) = \sum_{\substack{x,y\in\{0,1\}^{s_{i,i+1}}\\ w\in\{0,1\}}} \Big(\sum_{j=i+1}^{d} \tilde{\beta}(w,1)\tilde{\beta}((y_{s_{i,j}+1},\ldots,y_{s_{i,i+1}}),\vec{1})\\ \big(\tilde{add}_{i+1,j}(g,x,y_1,\cdots,y_{s_{i,j}})(\dot{V}'_{i,i+1}(x)+\dot{V}_{i,j}(y_1,\ldots,y_{s_{i,j}}))+\\ \tilde{mult}_{i+1,j}(g,x,y_1,\cdots,y_{s_{i,j}})\dot{V}'_{i,i+1}(x)\dot{V}_{i,j}(y_1,\ldots,y_{s_{i,j}})\big)\Big)+\\ \tilde{\beta}((x,y),\vec{1})Z_i(g)R_i(g_1,w)\end{aligned} \tag{11}$$

The equation holds because $\dot{V}_{i,j}$ agrees with $\tilde{V}_{i,j}$ on the Boolean hypercube $\{0,1\}^{s_{i,j}}$, as $Z_{i,j}(z) = 0$ for binary inputs.

Now $\mathcal{P}$ and $\mathcal{V}$ instead execute the zero knowledge sumcheck protocol on Equation 11. At the end of the protocol, $\mathcal{V}$ receives $\dot{V}_{i,i+1}(r^{(i,i+1)'}), \dot{V}_{i,i+1}(r^{(i,i+1)}), \ldots, \dot{V}_{i,d}(r^{(i,d)})$ for random points $r^{(i,i+1)'}, r^{(i,i+1)}, \ldots, r^{(i,d)}$ chosen by $\mathcal{V}$. They no longer leak information about $V_{i,i+1},\ldots,V_{i,d}$. $\mathcal{V}$ then evaluates $\tilde{mult}_{i,j}$ and $\tilde{add}_{i,j}$ on the randomness as before, computes $\tilde{\beta}((r^{(i,i+1)'}, r^{(i,i+1)}),\vec{1})$, $Z_i(g)$, $\tilde{\beta}(c,1)$ where $c \in \mathbb{F}$ is a random point chosen by $\mathcal{V}$ for the variable w. $\mathcal{V}$ also opens $R_i(g_1,w)$ at point c with $\mathcal{P}$ using zkPC, and checks that together with the points received from $\mathcal{P}$, they are consistent with the last message of the sumcheck, i.e., the oracle access to the evaluation of the polynomial in the zero knowledge sumcheck. $\mathcal{V}$ then uses these values to proceed to the second step of combining multiple evaluations, i.e., step 3(a) in Protocol 4. We have the following theorem.

Theorem D.4. *For every verifier* $\mathcal{V}^*$*, there exists a simulator* $\mathcal{S}$ *such that given oracle access to* $\dot{V}_i(g)$ *and* $\dot{V}_{i,i+1}(r^{(i,i+1)'}), \dot{V}_{i,i+1}(r^{(i,i+1)}), \ldots, \dot{V}_{i,d}(r^{(i,d)})$*,* $\mathcal{S}$ *is able to simulate the partial view of* $\mathcal{V}^*$ *in the zero knowledge sumcheck protocol on Equation 11.*

Proof sketch. The completeness and the soundness inherit from Protocol 5 and zkPC. For zero knowledge, we construct the simulator $\mathcal{S}$ by combining two simulators in Protocol 5 and in zkPC. Therefore, $\mathcal{V}$ only learns $\dot{V}_{i,i+1}(r^{(i,i+1)'}), \dot{V}_{i,i+1}(r^{(i,i+1)}), \ldots, \dot{V}_{i,d}(r^{(i,d)})$ at the end of the protocol, which leaks no information because of mask polynomials of $R_{i,i+1},\cdots,R_{i,d}$.

Efficiency. Compared with the plain sumcheck protocol in step 3(a) of Protocol 4, the prover costs extra $O(1)$ time to compute zkPC.Commit and zkPC.Open for $R_i(g_1,w)$ and $O(\log|C|)$ compute zkPC.Commit and zkPC.Open for the mask polynomial in Protocol 5. Therefore, the total prover time is still $O(|C|)$. The verifier time is $\min\{O(d\log|C| + d^2), O(|C|)\}$ while the proof size is also $\min\{O(d\log|C| + d^2), O(|C|)\}$.

Combine multiple evaluations in zero knowledge. With low degree extensions of $\dot{V}_{0,i},\ldots,\dot{V}_{i-1,i}$ and $\dot{V}_i$, we modify Equation 9

to

$$\begin{aligned}
&\sum_{k=0}^{i-1} \alpha_{k,i} \dot{V}_{k,i}(r^{(k,i)}) + \alpha'_{i-1,i} \dot{V}_{i-1,i}(r^{(i-1,i)'}) \\
&= \sum_{k=0}^{i-1} \left(\alpha_{k,i} \sum_{x \in \{0,1\}^{s_i}} \tilde{C}_{k,i}(r^{(k,i)}, x) \dot{V}_i(x) \right) + \\
&\alpha'_{i-1,i} \sum_{x \in \{0,1\}^{s_i}} \tilde{C}_{i-1,i}(r^{(i-1,i)'}, x) \dot{V}_i(x) + \\
&\sum_{k=0}^{i-1} \alpha_{k,i} Z_{k,i}(r^{(k,i)}) \sum_{w \in \{0,1\}} R_{k,i}(r_1^{(k,i)}, w) + \\
&\alpha'_{i-1,i} Z_{i-1,i}(r^{(i-1,i)'}) \sum_{w \in \{0,1\}} R_{i-1,i}(r_1^{(i-1,i)'}, w) \qquad (12) \\
&= \sum_{x \in \{0,1\}^{s_i}, w \in \{0,1\}} \Bigg[\tilde{\beta}(w,1) \dot{V}_i(x) \\
&\left(\sum_{k=0}^{i-1} \alpha_{k,i} \tilde{C}_{k,i}(r^{(k,i)}, x) + \alpha'_{i-1,i} \tilde{C}_{i-1,i}(r^{(i-1,i)'}, x) \right) \\
&\tilde{\beta}(x, \vec{1}) \sum_{k=0}^{i-1} \alpha_{k,i} Z_{k,i}(r^{(k,i)}) R_{k,i}(r_1^{(k,i)}, w) + \\
&\tilde{\beta}(x, \vec{1}) \alpha'_{i-1,i} Z_{i-1,i}(r^{(i-1,i)'}) R_{i-1,i}(r_1^{(i-1,i)'}, w) \Bigg],
\end{aligned}$$

The equation holds because $\dot{V}_i$ agrees with $\tilde{V}_i$ on the Boolean hypercube $\{0,1\}^{s_i}$, as $Z_i(z) = 0$ for binary inputs. To execute the second step, the prover commits to mask polynomials of $R_{0,i}, \ldots, R_{i-1,i}$ using zkPC. $\mathcal{P}$ and $\mathcal{V}$ then run the zero knowledge sumcheck protocol on Equation 12. At the end of the protocol, the verifier receives evaluations of $R_{0,i}(r_1^{(0,i)}, c), \ldots, R_{i-1,i}(r_1^{(i-1,i)}, c)$ on a random point c chosen by $\mathcal{V}$ for the variable w. He opens $R_{0,i}(r_1^{(0,i)}, c)$, $\ldots, R_{i-1,i}(r_1^{(i-1,i)}, c)$ using the zkPC. Then $\mathcal{V}$ evaluates $g_i(r^{(i)})$ as before, computes all $Z_{k,i}(r^{(k,i)})$, $Z_{i-1,i}(r^{(i-1,i)'})$, $\beta(c,1)$, $\beta(r^{(i)}, \vec{1})$, shaves them off to obtain the evaluation of $\dot{V}_i(r^{(i)})$. We have the following theorem.

Theorem D.5. *For every verifier $\mathcal{V}^*$, there exists a simulator $\mathcal{S}$ such that given oracle access to $\dot{V}_i(r^{(i)})$ and $\dot{V}_{0,i}(r^{(0,i)})$, $\ldots$, $\dot{V}_{i-1,i}(r^{(i-1,i)})$, $\dot{V}_{i-1,i}(r^{(i-1,i)'})$, $\mathcal{S}$ is able to simulate the partial view of $\mathcal{V}^*$ in the zero knowledge sumcheck protocol on Equation 12.*

Proof sketch. The completeness and the soundness inherits from the zero knowledge sumcheck protocol and zkPC. For zero knowledge, we combine the simulator $\mathcal{S}_1$ in zkPC and the simulator $\mathcal{S}_2$ in the zero knowledge sumcheck protocol to construct the simulator $\mathcal{S}$. Therefore, $\mathcal{V}$ only learns $\dot{V}_i(r^{(i)})$ at the end of the protocol, which leaks no information about $\tilde{V}_i$ because of the mask polynomail of R_i.

Efficiency. Compared with the plain sumcheck protocol in the second step, the prover costs extra $O(i)$ time to compute zkPC.Commit and zkPC.Open for i constant size polynomials of $R_{0,i}, \ldots, R_{i-1,i}$ and $O(\log |C|)$ compute zkPC.Commit and zkPC.Open for the mask polynomial in Protocol 5. Therefore, the total prover time is $O(|C| + 1 + \ldots + d) = O(|C|)$. The verifier time is $\min\{O(d \log |C| + d^2), O(|C|)\}$ while the proof size is also $\min\{O(d \log |C| + d^2), O(|C|)\}$.

D.4 Putting Everything Together

Combining the zero knowledge variants of step 3(a) and 3(b) in Protocol 4 with the zkPC scheme, we get a zero knowledge argument protocol for general arithmetic circuits.

Proof Sketch of Theorem 4.1. The correctness and the soundness follow from those of the three building blocks, by Theorem D.4, D.5 and Definition D.2.

To prove zero knowledge, consider a simulator $\mathcal{S}$ that calls the simulator $\mathcal{S}_1$ of zero knowledge sumcheck given in Theorem D.4 for step 1, the simulator $\mathcal{S}_2$ of combining multiply claims to one claim with zero knowledge given in Theorem D.5 for step 2 and the simulator $\mathcal{S}_3$ of zkPC in Definition D.2 for committing and opening of all hiding polynomials as subroutines. Then $\mathcal{S}$ can simulate the partial view of every verifier $\mathcal{V}^*$ for any general arithmetic circuit C only given oracle access to x.

The complexity of our zero knowledge argument scheme follows from the efficency of Protocol 4 and the extra complexity of applying zkPC.Commit to the input layer demonstrated in [50].