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

Jiaheng Zhang* Weijie Wang† Yinuo Zhang* Yupeng Zhang‡

Abstract

We propose a new doubly efficient interactive proof protocol for general arithmetic circuits. The protocol generalizes the doubly efficient interactive proof for layered circuits proposed by Goldwasser, Kalai and Rothblum to arbitrary circuits, while preserving the optimal prover complexity that is strictly linear to 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.

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 1 second to generate the proof for a circuit with 600,000 gates, which is 7 times faster than the original interactive proof protocol on the corresponding layered circuit. The proof size is 229 kilobytes and the verifier time is 0.56 second. Our implementation can take general arithmetic circuits generated by existing tools directly, without transforming them to layered circuits with high overhead on the size of the circuits.

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 it was introduced by Goldwasser, Micali, and Rackoff [10] 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 [11, 15] and MIP=NEXP [3].

In recent years, there is a great progress on turning interactive proofs from purely theoretical constructions to practical schemes with efficient implementations. In the seminal work of [9], 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 polynomial on the size of the circuit, and the time to validate the result (verifier time) is close to linear to 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 to the size of the circuit and linear to the depth of the circuit, which is *succinct*

*University of California, Berkeley. **Email:** {jiaheng_zhang,yinuo}@berkeley.edu.
†Shanghai Jiao Tong University. **Email:** wangnick@sjtu.edu.cn.
‡Texas A&M University. **Email:** zhangyp@tamu.edu.

for bounded-depth circuits. We refer the protocol in [9] as the *GKR* protocol in this paper. 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 [8], and then to close to linear for various circuits with different structures [16, 17, 24]. Finally, in [20], Xie et al. proposed an algorithm to improve the prover time to strictly linear ($O(|C|)$) for arbitrary layered arithmetic circuits without assuming any structures, which is optimal asymptotically and very efficient in practice.

Another important advance of interactive proofs recently is using them to construct efficient zero knowledge argument schemes. In [22], 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 the witness directly to the verifier. Following the framework, there are many subsequent zero knowledge argument constructions based on interactive proofs, including [14, 18, 20, 21, 23]. 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 remove the restriction of the GKR protocol on supporting only layered circuits, without introducing any overhead on the prover time?

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 strictly linear to 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 for the first time with no asymptotic overhead. For a general circuit of size $|C|$ and depth d, the prover time is $O(|C|)$ and the proof size is $O(d\log^2|C|+d^2)$. For structured circuits, the verifier time is $O(d^2\log|C|+d\log^2|C|)$. In the second variant of the protocol, we can reduce the proof size to $O(d\log|C|+d^2)$, while the verifier is $O(|C|)$ regardless of the structure of the circuit. See Section 3.

- Together with zero knowledge polynomial commitments, we construct zero knowledge arguments for general arithmetic circuits. The zero knowledge version of our interactive proof protocols only incurs a small overhead of $O(d^2\log d)$ on the prover time, and preserves the same proof size and the verifier time. See Section 4.

- We fully implement a system, **virgo++**, for our new interactive proof protocols and zero knowledge arguments. We show that on circuits with $d=50$ and $d=75$, our new protocols are 2-8× 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 2.5–5× more than the original GKR protocol on layered circuits. Therefore, as long as padding the general circuit to layered circuit makes it more than 2.5–5× larger, our new protocol will have faster prover time. See Section 5.

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 $i+1$. Then starting from the output layer (layer 0), $\mathcal{P}$ and $\mathcal{V}$ reduce the correctness of the values in layer i to the correctness of the values in layer $i+1$ recursively, and eventually to the correctness of the input. $\mathcal{V}$ can then validates the correctness of the input on her own, which completes the reduction and guarantees that the output is correctly computed. To do so, we use the multilinear extension $\tilde{V}_i()$ of the i-th layer, which is a multilinear polynomial that agrees with all the values in the i-th layer on the Boolean hypercube, i.e., $\tilde{V}_i(0,0,\ldots,0) = \mathbf{V}_i[0], \tilde{V}_i(0,0,\ldots,1) = \mathbf{V}_i[1], \ldots$ where $\mathbf{V}_i$ is the array representing the values in the i-th layer of the circuit. Assuming for simplicity that all layers have S gates and $\tilde{V}$ takes $s = \log S$ variables, we can write $\tilde{V}_i()$ as a equation of $\tilde{V}_{i+1}()$:

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

for all $z \in \{0,1\}^s$, where $\tilde{add}_{i+1}(z,x,y)$ and $\tilde{mult}_{i+1}(z,x,y)$ are polynomials describing the addition/multiplication gates and their connections in the circuit between layer i and layer $i+1$. With this equation, the GKR protocol invokes the sumcheck protocol (See Section 2.2), which reduces the correctness of $\tilde{V}_i(g)$ at a random point $g \in \mathbb{F}^s$ to the correctness of $\tilde{V}_{i+1}(u)$ and $\tilde{V}_{i+1}(v)$ at two random points $u, v \in \mathbb{F}^s$. Then $\tilde{V}_{i+1}(u)$ and $\tilde{V}_{i+1}(v)$ can be combined back to a single evaluation of $\tilde{V}_{i+1}(w)$ for $w \in \mathbb{F}^s$. At this point, $\tilde{V}_{i+1}(w)$ can be further reduced to an evaluation of $\tilde{V}_{i+2}$ 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 $O(S)$ in each layer [20] and the proof size is only $O(\log S)$. Therefore, the total prover time is $O(dS) = O(|C|)$ and the proof size is $O(d \log S) = O(d \log |C|)$.

Extending GKR to general circuits naively. The nice equation above relies on the fact that gates in layer i can only take input from gates in layer $i+1$. In a general circuit, a gate in layer i can take input from any gate in layer j for $j > i$. As circuits cannot contain cycles (otherwise we cannot evaluate the circuit), we can still assign a layer number to each gate in the topological order. Thus a gate can take input from any gate in layers above, but not below. More interestingly, the gate in layer i has to have at least one input from layer $i+1$, otherwise it cannot belong to layer i in the topological order. Because of this generalization, we can write $\tilde{V}_i()$ as:

$$\begin{aligned}\tilde{V}_i(z) = \sum_{x,y\in\{0,1\}^s} &(\tilde{add}_{i+1,i+1}(z,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + \tilde{mult}_{i+1,i+1}(z,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)) \\ &+ \tilde{add}_{i+1,i+2}(z,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+2}(y)) + \tilde{mult}_{i+1,i+2}(z,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+2}(y)) \\ &+ \ldots + \tilde{add}_{i+1,d}(z,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_d(y)) + \tilde{mult}_{i+1,d}(z,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_d(y))).\end{aligned}$$

Namely, we have multiple parts in the summation, one for each layer $j = i+1, i+2, \ldots, d$. $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck protocol on this equation, which reduces the correctness of $\tilde{V}_i(g)$ at a random point $g \in \mathbb{F}^s$ to the correctness of $\tilde{V}_{i+1}(u)$ and $\tilde{V}_{i+1}(v), \tilde{V}_{i+2}(v), \ldots, \tilde{V}_d(v)$ at random points $u, v \in \mathbb{F}^s$. Moreover, when reaching layer $i+1$, now $\mathcal{V}$ has many evaluations about $\tilde{V}_{i+1}$ instead

of just two. In the sumcheck protocols of all layers below, $\mathcal{V}$ has received one evaluation of $\tilde{V}i+1$ from the sumcheck of layer $k = 0, \ldots, i-1$, and two evaluations from layer i. Nevertheless, $\mathcal{V}$ can combine all these evaluations into one evaluation $\tilde{V}_{i+1}(w)$ using the original protocol multiple times with $\mathcal{P}$. $\mathcal{P}$ and $\mathcal{V}$ can then run the protocol recursively layer by layer just as the original GKR protocol to reduce the correctness of the output layer to the input layer.

It is not hard to show that the generalized protocol is secure. However, it introduces a big overhead on the prover time. The size of all the polynomials in the generalized equation becomes $O((d-i)S)$, and the total prover time for the sumcheck protocol of all layers becomes $O(dS + (d-1)S + \ldots + S) = O(d^2S) = O(d|C|)$. This is as bad as padding the general circuit to a layered circuit and running the original GKR protocol. Even worse, the second step of combining multiple evaluations into one also introduces a prover time of $O(d|C|)$, simply because there are now $i+1$ evaluations to combine instead of two.

Extending GKR to general circuits with optimal prover time. In order to preserve the linear prover time, we introduce two new techniques. First, we observe that the key reason why the prover time of the sumcheck protocol on the generalized equation becomes $O((d-i)S)$ is that the multilinear extension $\tilde{V}_j$ of the entire layer j for $j > i$ is used. As layer j has S gates and its multilinear extension is uniquely defined by these gates, merely writing out all the polynomials $\tilde{V}_j$ for $j > i$ takes $O((d-i)S)$ time. There is no hope to reduce the prover time if we define the equation in this way. Meanwhile, it is also not necessary to use all the gates in layers above, because gates in layer i can at most take input from $2S$ gates in total. Therefore, we propose a new equation to write $\tilde{V}_i$ as a function of multilinear extensions define by *only those values used by layer i* from layer $j > i$. In particular, we have

$$\begin{aligned}\tilde{V}_i(z) = \sum_{x,y\in\{0,1\}^{s'}} & (\tilde{add}_{i+1,i+1}(z,x,y)(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,i+1}(y)) + \tilde{mult}_{i+1,i+1}(z,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+1}(y)) \\ & + \tilde{add}_{i+1,i+2}(z,x,y)(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,i+2}(y)) + \tilde{mult}_{i+1,i+2}(z,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+2}(y)) \\ & + \ldots + \tilde{add}_{i+1,d}(z,x,y)(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,d}(y)) + \tilde{mult}_{i+1,d}(z,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,d}(y))),\end{aligned}$$

where $\tilde{V}_{i,j}$ is the multilinear extension of values used by layer i from layer j for $j > i$ arranged in a pre-defined order,i.e., a subset of the values in the entire layer j. Now the total size of the all these polynomials is bounded by $2S$. We also change the range of the summation from $\{0,1\}^s$ to $\{0,1\}^{s'}$ to informally denote that now the number of variables is smaller. We will show how to deal with different sizes from different layers in our formal protocols. We then design a new algorithm for the prover to run the sumcheck protocol on the equation above with time complexity $O(S)$ by utilizing the sparsity of the polynomials $\tilde{add}$ and $\tilde{mult}$. The formal algorithms are presented in Section 3.2.

Combining evaluations of different multilinear extensions. At the end of the sumcheck protocol on the equation above, $\mathcal{P}$ and $\mathcal{V}$ reduce the correctness of $\tilde{V}_i(g)$ at a random point $g \in \mathbb{F}^s$ to the correctness of $\tilde{V}_{i,i+1}(u)$ and $\tilde{V}_{i,i+1}(v), \tilde{V}_{i,i+2}(v), \ldots, \tilde{V}_{i,d}(v)$ at random points $u, v \in \mathbb{F}^{s'}$. When reaching layer $i+1$, $\mathcal{V}$ has many evaluations of multilinear extensions of *subsets* of $\mathbf{V}_{i+1}$. Now we cannot even use the original protocol to combine these points into one, as they are evaluations of *different* multilinear extensions, not mentioning that we want to reduce the complexity of the prover time in this step. Our second technique is to compute them using a layered arithmetic circuit and reduce these evaluations to a single evaluation of $\tilde{V}_i$ through the original GKR protocol. At this point, the random points in these evaluations are already fixed by the verifier. We construct a circuit whose input is the values $\mathbf{V}_{i+1}$ of the entire layer $i+1$, and all the random points in the evaluations,

denoted as $v^{(0)}, v^{(1)}, \ldots, v^{(i)}$ and $u^{(i)}$. The output of the circuit is exactly the evaluations of the multilinear extensions of the subsets, received from the sumcheck protocols for all layers below, i.e., $\tilde{V}_{0,i+1}(v^{(0)}), \tilde{V}_{1,i+1}(v^{(1)}), \ldots, \tilde{V}_{i,i+1}(v^{(i)})$ and $\tilde{V}_{i,i+1}(u^{(i)})$. To compute the output, the circuit selects all the subsets from input $\mathbf{V}_i$ and arrange them in the predefined order, which can be determined by the structure of the general circuit. The circuit then evaluates the multilinear extensions defined by these subsets at points from input $v^{(0)}, v^{(1)}, \ldots, v^{(i)}$ and $u^{(i)}$. By executing the original GKR protocol on this circuit, $\mathcal{P}$ and $\mathcal{V}$ reduce the correctness of $\tilde{V}_{0,i+1}(v^{(0)}), \tilde{V}_{1,i+1}(v^{(1)}), \ldots, \tilde{V}_{i,i+1}(v^{(i)})$ and $\tilde{V}_{i,i+1}(u^{(i)})$ to a single evaluation of the input. As part of the input $v^{(0)}, v^{(1)}, \ldots, v^{(i)}$ and $u^{(i)}$ is known to the verifier, it is easy to subtract it from the evaluation and obtain $\tilde{V}_i(w)$, a single evaluation of the multilinear extension $\tilde{V}_i$ at a random point $w \in \mathbb{F}^s$. With this single evaluation, $\mathcal{P}$ and $\mathcal{V}$ can continue the sumcheck for layer $i+1$ recursively and proceed all the way to the input layer. 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 1 and Section 3.3 for the design of the circuit and the details of the protocol.

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 $O(|C|)$.

Building zero knowledge arguments. Following the framework of [20, 21], we build zero knowledge arguments for general arithmetic circuits using our new protocol. The prover first commits to the witness using a zero knowledge polynomial commitment scheme. Then after the verifier receives the output, $\mathcal{P}$ and $\mathcal{V}$ run the zero knowledge sumcheck protocol instead of the plain sumcheck on the equation above. We use the zero knowledge sumcheck in [20] as is, and there is no leakage of the values in the circuit during the sumcheck. At the end of the sumcheck, to eliminate the leakage of the evaluations of the multilinear extensions, we instead use the low degree extensions such that revealing one evaluation leaks no information about the underlying function, and thus the values in the circuit. We then combine these low degree extensions using a modified circuit and run the zero knowledge version of the orginal GKR proposed in [20] to reduce them to the low degree extension of the values in layer $i+1$. To preserve the complexity of our plain protocol without zero knowledge, we introduce several optimizations to simplify the masking polynomials in the low degree extensions, relying on the fact that only one evaluation of each low degree extensions is revealed to the verifier, and the low degree extensions of the subsets are not used in the sumcheck of the next layer.

1.3 Related Work

Interactive proofs were formally defined by Goldwasser, Micali, and Rackoff in [10]. In the seminal work of [9], 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 [8]. Several follow-up papers further reduce the prover time for circuits with special structures. Justin Thaler [16] introduced a protocol with $O(|C|)$ prover time for regular circuits where the wiring pattern can be described in constant space and time. In the same work, a protocol with prove time $O(|C| \log |C'|)$ was proposed for data parallel circuits with many copies of small circuits of size $|C'|$. The complexity was further improved to $O(|C| + |C'| \log |C'|)$ by Wahby et al. in [17]. For circuits with many non-connected but different copies, Zhang et al. [24] showed a protocol with $O(|C| \log |C'|)$ prover time. Eventually, Xie et al. [20] proposed a variant of the GKR protocol with $O(|C|)$ prover time for

arbitrary layered arithmetic circuit. 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 [22], Zhang et al. extended the GKR protocol to an argument system using polynomial commitments. Subsequent works [14, 18, 20, 21, 23] followed the framework and constructed efficient zero knowledge argument schemes based on interactive proofs. We follow the approach of [7, 20, 21] and constructs zero knowledge arguments for general circuits instead of layered circuits. Notably, there is a recent work [14] on constructing interactive proof-based zero knowledge argument 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, which can be used for delegation of computations and constructing zero knowledge arguments with polynomial commitments only on the witness. Therefore, though the protocol in [14] also uses the sumcheck protocol and polynomial commitments, the challenges and the techniques are very different from our protocols.

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) [12], interactive oracle proofs (IOP) [5], discrete-log [6], MPC-in-the-head [2] and lattice [4]. We refer the readers to surveys [19] and recent papers [21] 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 1. *Let C be a function. A pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an interactive proof for f 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 (verifier time) if the total communication (verifier time) is $O(\mathsf{polylog}(|C|, |x|))$.

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 $2 \leq i \leq \ell - 1$, $\mathcal{P}$ 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.

2.2 Doubly Efficient Interactive Proofs for Layered Arithmetic Circuits

In [9], 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

$$\sum\nolimits_{b_1,b_2,\ldots,b_\ell \in \{0,1\}} f(b_1, b_2, ..., b_\ell).$$

Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \ldots, b_\ell$. Lund et al. [11] proposed a *sumcheck* protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ that H is the correct sum. We provide a description of the sumcheck protocol in Protocol 1. The proof size of the sumcheck protocol is $O(\tau\ell)$, where τ is the variable-degree of f, as in each round, $\mathcal{P}$ sends a univariate polynomial of 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 (**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 $x_1, x_2, \ldots, x_\ell \in \{0,1\}^\ell$. $\tilde{V}$ *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 $\forall i \in [0, n-1], A(i_1, \ldots, i_{\log n}) = a_i$. Therefore, in this paper, we abuse the use of multilinear extension on an array as the multilinear extension $\tilde{A}$ of A.

2.2.2 GKR Protocol

Using the sumcheck protocol as a building block, Goldwasser et al. [9] 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 $\mathcal{V}$, or is given as an oracle access. If the check passes, $\mathcal{V}$ accepts the claimed output.

Notation. We follow the convention in prior works of GKR protocols [8, 16, 20–22]. 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 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 definitions, V_i can be written as follows:

$$\begin{aligned} V_i(z) = \sum_{x,y\in\{0,1\}^{s_{i+1}}} & (add_{i+1}(z,x,y)(V_{i+1}(x) + V_{i+1}(y)) \\ & + mult_{i+1}(z,x,y)(V_{i+1}(x)V_{i+1}(y))) \end{aligned} \tag{1}$$

for any $z \in \{0,1\}^{s_i}$.

In the equation above, V_i is expressed as a summation, so $\mathcal{V}$ can use the sumcheck protocol to check that it is computed correctly. As the sumcheck protocol operates on polynomials defined on $\mathbb{F}$, we rewrite the equation with their multilinear extensions:

$$\begin{aligned} \tilde{V}_i(g) &= \sum_{x,y\in\{0,1\}^{s_{i+1}}} f_i(g,x,y) \\ &= \sum_{x,y\in\{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ &\quad + \tilde{mult}_{i+1}(g,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))), \end{aligned} \tag{2}$$

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

Protocol. With Equation 2, 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 2 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, but 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 claims and the sumcheck protocols would increase exponentially in d.

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 [7]. 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 2, this random linear combination can be written as

$$
\begin{aligned}
&\alpha_{i,1}\tilde{V}_i(u) + \alpha_{i,2}\tilde{V}_i(v) \\
=&\alpha_{i,1} \sum_{x,y\in\{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(u,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + \tilde{mult}_{i+1}(u,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))) \\
+&\alpha_{i,2} \sum_{x,y\in\{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(v,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) + \tilde{mult}_{i+1}(v,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))) \\
=& \sum_{x,y\in\{0,1\}^{s_{i+1}}} ((\alpha_{i,1}\tilde{add}_{i+1}(u,x,y) + \alpha_{i,2}\tilde{add}_{i+1}(v,x,y))(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\
&+ (\alpha_{i,1}\tilde{mult}_{i+1}(u,x,y) + \alpha_{i,2}\tilde{mult}_{i+1}(v,x,y))(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))) \qquad (3)
\end{aligned}
$$

$\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 3 instead of Equation 2. 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 an layer above recursively until the input layer.

The formal GKR protocol is presented in Protocol 2. With the optimal algorithms generating the proof in linear time to the number of gates in the circuit proposed in [20], we have the following theorem:

Theorem 1. *[20]. Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a depth-d 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 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 $O(n+k+d \log |C|+T)$.*

Protocol 2. *Let $\mathbb{F}$ be a prime field. Let C: $\mathbb{F}^n \to \mathbb{F}^k$ be a d-depth layered arithmetic circuit. $\mathcal{P}$ wants to convince that **out** $= C($**in**$)$ where **in** is the input from $\mathcal{V}$, and **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 **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*

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

   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 $\tilde{mult}_1(g^{(0)}, u^{(1)}, v^{(1)})\tilde{V}_1(u^{(1)})\tilde{V}_1(v^{(1)}) + \tilde{add}_1(g^{(0)}, u^{(1)}, v^{(1)})(\tilde{V}_1(u^{(1)}) + \tilde{V}_1(v^{(1)}))$ equals to the last message of the sumcheck.
3. *For $i = 1, ..., d-1$:*
   - *$\mathcal{V}$ 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*

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

   - *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 `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* `1`*; otherwise, output* `0`*.*

3 Generalizing GKR to Arbitrary Arithmetic Circuits

Though the GKR protocol has great prover efficiency as demonstrated in [20, 21] 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. On one hand, it introduces extra overhead to transform a general circuit to a layered circuit. In the worst case, by relaying the values layer by layer, the size of the circuit blows up quadratically (from $O(|C|)$ to $O(d|C|)$ for a circuit of size $|C|$ and depth d). On the other hand, it is also inconvenient to implement circuits in a strictly layered way, and existing tools such as rank-1-constraint-system (R1CS) cannot be used directly. Therefore, 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 based on the topological order. In particular, if gate g is not an input, suppose gate u and gate v are the input gates of g, then $\mathsf{layer}(g) = \min(\mathsf{layer}(u), \mathsf{layer}(v)) - 1$, where the function $\mathsf{layer}(x)$ represents the layer of the gate x. 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 can only take input from gates with larger layer numbers.

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 natually 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. We still use $\tilde{f}$ to represent the multilinear extensions of the function f.

3.1 A Naive Generalization of the GKR Protocol

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 2 to have one *add* and one *mult* for each layer above. For each gate in layer i, we assume the left input gate is from layer $i+1$, and rewrite the sumcheck equation in Equation 2 as:

$$\begin{aligned}\tilde{V}_i(g) = \sum_{x\in\{0,1\}^{s_{i+1}}} (& \sum_{y\in\{0,1\}^{s_{i+1}}} \tilde{add}_{i+1,i+1}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ & + \sum_{y\in\{0,1\}^{s_{i+2}}} \tilde{add}_{i+1,i+2}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+2}(y)) + \ldots + \sum_{y\in\{0,1\}^{s_d}} \tilde{add}_{i+1,d}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_d(y)) \\ & + \sum_{y\in\{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1,i+1}(g,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)) + \sum_{y\in\{0,1\}^{s_{i+2}}} \tilde{mult}_{i+1,i+2}(g,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+2}(y)) \\ & + \ldots + \sum_{y\in\{0,1\}^{s_d}} \tilde{mult}_{i+1,d}(g,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_d(y)))\end{aligned} \tag{4}$$

for any $g \in \mathbb{F}^{s_i}$. With this equation, starting from the output, in round i, the first step is that $\mathcal{P}$ and $\mathcal{V}$ engage the sumcheck protocol on Equation 4 to reduce one claim about layer i to claims about previous layers. At the end of 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, and $\mathcal{V}$ evaluates all *add* and *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). Denote the randomness of these claims as $g^{(0)}, g^{(1)}, \ldots, g^{(i)}$. $\mathcal{V}$ picks a random number $\alpha_{i,j}$ for each claim, and we can rewrite Equation 4

[1] Note that as we support non-layered 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.

as:

$$
\begin{aligned}
&\alpha_{i,0}\tilde{V}_i(g^{(0)}) + \alpha_{i,1}\tilde{V}_i(g^{(1)}) + \ldots + \alpha_{i,i}\tilde{V}_i(g^{(i)}) \\
&= \sum_{j=0}^{i} \alpha_{i,j} \Big(\sum_{x\in\{0,1\}^{s_{i+1}}} \Big(\sum_{y\in\{0,1\}^{s_{i+1}}} \tilde{add}_{i+1,i+1}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)+\tilde{V}_{i+1}(y)) \\
&+ \sum_{y\in\{0,1\}^{s_{i+2}}} \tilde{add}_{i+1,i+2}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)+\tilde{V}_{i+2}(y)) + \ldots + \sum_{y\in\{0,1\}^{s_d}} \tilde{add}_{i+1,d}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)+\tilde{V}_d(y)) \\
&+ \sum_{y\in\{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1,i+1}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)) + \sum_{y\in\{0,1\}^{s_{i+2}}} \tilde{mult}_{i+1,i+2}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+2}(y)) \\
&+ \ldots + \sum_{y\in\{0,1\}^{s_d}} \tilde{mult}_{i+1,d}(g^{(j)},x,y)(\tilde{V}_{i+1}(x)\tilde{V}_d(y)))\Big)
\end{aligned}
\tag{5}
$$

$\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 5 instead of Equation 4. 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 until the input layer.

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. In particular, (1) in the first step of the sumcheck protocol on Equation 4, 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 takes $O(S_{i+1}+S_{i+2}+\ldots+S_d)$ time, not mentioning 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. (2) In the second step of random linear combinations, as shown in Equation 5, for layer i, $\mathcal{V}$ combines $i+1$ claims and on the right side of the equation each $\tilde{add}$ and $\tilde{mult}$ has to be evaluated on $i+1$ different random points $g^{(j)}$. This again introduces a prover time of $O(d|C|)$ for the second step. 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.

3.2 Sumcheck with Linear Prover Time

As explained above, the main overhead of the sumcheck on Equation 4 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 instead use 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 of used gates instead of the labels of the the entire layers. In particular, $add_{i,j}(z,x,y) = 1$ ($multi_{i,j}(z,x,y) = 1$) 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 these definitions,

by taking their multilinear extensions, we can rewrite Equation 4 as

$$\begin{aligned}\tilde{V}_i(g) = &\sum_{x\in\{0,1\}^{s_{i,i+1}}} \Big(\sum_{y\in\{0,1\}^{s_{i,i+1}}} \tilde{add}_{i+1,i+1}(g,x,y)(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,i+1}(y)) \\ &+ \sum_{y\in\{0,1\}^{s_{i,i+2}}} \tilde{add}_{i+1,i+2}(g,x,y)(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,i+2}(y)) + \ldots + \sum_{y\in\{0,1\}^{s_{i,d}}} \tilde{add}_{i+1,d}(g,x,y)(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,d}(y)) \\ &+ \sum_{y\in\{0,1\}^{s_{i,i+1}}} \tilde{mult}_{i+1,i+1}(g,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+1}(y)) + \sum_{y\in\{0,1\}^{s_{i,i+2}}} \tilde{mult}_{i+1,i+2}(g,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+2}(y)) \\ &+ \ldots + \sum_{y\in\{0,1\}^{s_{i,d}}} \tilde{mult}_{i+1,d}(g,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,d}(y))\Big)\end{aligned} \tag{6}$$

In Equation 6, 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 $O(S_i)$. Therefore, using similar ideas proposed in [20], 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 6 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 asymptotic overhead. We combine them into a single sum in the following way. Without loss of generality, we suppose $s_{i,i+1}$ is the largest. We can then rewrite Equation 6 as:

$$\begin{aligned}\tilde{V}_i(g) = &\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{add}_{i+1,i+1}(g,x,y)(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,i+1}(y_1,\ldots,y_{s_{i,i+1}})) \\ &+ y_{s_{i,i+2}+1}\cdot\ldots\cdot y_{s_{i,i+1}}\tilde{add}_{i+1,i+2}(g,x,y_1,y_2,\ldots,y_{s_{i,i+2}})(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,i+2}(y_1,y_2,\ldots,y_{s_{i,i+2}}))+\ldots \\ &+ y_{s_{i,d}+1}\cdot\ldots\cdot y_{s_{i,i+1}}\tilde{add}_{i+1,d}(g,x,y_1,y_2,\ldots,y_{s_{i,d}})(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}})) \\ &+ \tilde{mult}_{i+1,i+1}(g,x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+1}(y_1,\ldots,y_{s_{i,i+1}})) \\ &+ y_{s_{i,i+2}+1}\cdot\ldots\cdot y_{s_{i,i+1}}\tilde{mult}_{i+1,i+2}(g,x,y_1,y_2,\ldots,y_{s_{i,i+2}})(\tilde{V}_{i,i+1}(x)\cdot\tilde{V}_{i,i+2}(y_1,y_2,\ldots,y_{s_{i,i+2}}))+\ldots \\ &+ y_{s_{i,d}+1}\cdot\ldots\cdot y_{s_{i,i+1}}\tilde{mult}_{i+1,d}(g,x,y_1,y_2,\ldots,y_{s_{i,d}})(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}}))\end{aligned} \tag{7}$$

Note that the only difference between Equation 6 and 7 is that in Equation 7 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 takes $y_1,\ldots,y_{s_{i,j}}$, we multiply each term with $y_{s_{i,j}+1}\cdot y_{s_{i,j}+2}\cdot\ldots\cdot y_{s_{i,i+1}}$. This guarantees that the term only appears once in the sum, when $y_{s_{i,j}+1}=y_{s_{i,j}+2}=\ldots=y_{s_{i,i+1}}=1$. Therefore, Equation 7 holds. In fact, $y_{s_{i,j}+1}\cdot y_{s_{i,j}+2}\cdot\ldots\cdot y_{s_{i,i+1}}$ is exactly the identity polynomial $\beta(\vec{1},(y_{s_{i,j}+1},y_{s_{i,j}+2},\ldots,y_{s_{i,i+1}}))$ where $\beta(x,y)=1$ iff $x=y$, which is commonly used in the literature to combine multiple sums [7]. 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.

Next, we present an algorithm for $\mathcal{P}$ to run the sumcheck protocol on Equation 7 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.

3.2.1 Linear-time sumcheck for products of multilinear functions [16]

In [16], 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 2. Algorithm 2 invokes Algorithm 1 FunctionEvaluations() as a subroutine. The

Algorithm 1 $\mathcal{F} \leftarrow \mathsf{FunctionEvaluations}(f, \mathbf{A}, r_1, \ldots, r_\ell)$

Input: Multilinear f on ℓ variables, initial bookkeeping table $\mathbf{A}$, random $r_1, \ldots, r_\ell$;
Output: All function evaluations $f(r_1, \ldots, r_{i-1}, t, b_{i+1}, \ldots, b_\ell)$;

1: **for** $i = 1, \ldots, \ell$ **do**
2: **for** $b \in \{0,1\}^{\ell-i}$ **do** *// b is both a number and its binary representation.*
3: **for** $t = 0, 1, 2$ **do**
4: Let $f(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}[b] \cdot (1-t) + \mathbf{A}[b + 2^{\ell-i}] \cdot t$
5: $\mathbf{A}[b] = \mathbf{A}[b] \cdot (1 - r_i) + \mathbf{A}[b + 2^{\ell-i}] \cdot r_i$
6: Let $\mathcal{F}$ contain all function evaluations $f(.)$ computed at Step 4
7: **return** $\mathcal{F}$

Algorithm 2 $\{a_1, \ldots, a_\ell\} \leftarrow \mathsf{SumCheckProduct}(f, \mathbf{A}_f, g, \mathbf{A}_g, r_1, \ldots, r_\ell)$

Input: Multilinear f and g, initial bookkeeping tables $\mathbf{A}_f$ and $\mathbf{A}_g$, random $r_1, \ldots, r_\ell$;
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});

1: $\mathcal{F} \leftarrow \mathsf{FunctionEvaluations}(f, \mathbf{A}_f, r_1, \ldots, r_\ell)$
2: $\mathcal{G} \leftarrow \mathsf{FunctionEvaluations}(g, \mathbf{A}_g, r_1, \ldots, r_\ell)$
3: **for** $i = 1, \ldots, \ell$ **do**
4: **for** $t \in \{0, 1, 2\}$ **do**
5: $a_{it} = \sum_{b \in \{0,1\}^{\ell-i}} f(r_1, \ldots, r_{i-1}, t, b) \cdot g(r_1, \ldots, r_{i-1}, t, b)$ *// All evaluations needed are in $\mathcal{F}$ and $\mathcal{G}$.*
6: **return** $\{a_1, \ldots, a_\ell\}$;

algorithms are exactly the same as Algorithm 1 and 3 in [20]. Both Algorithm 1 and Algorithm 2 run in time $O(2^\ell)$ and the formal proof can be found in [16, 20]. We have a lemma as follows:

Lemma 1. *Given multilinear functions f and g on ℓ variables and a* bookkeeping table $\boldsymbol{A}_f$ *for f and a* bookkeeping table $\boldsymbol{A}_g$ *for g, 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, respectively.*

3.2.2 Linear-time sumcheck for Equation 7

The idea of the prover algorithm is similar to that proposed in [20]. The algorithm proceeds in two phases, one summing x and the other summing y. For the ease of presentation, let us consider the sumcheck on a particular class of equations:

$$\begin{aligned}\sum_{x,y \in \{0,1\}^\ell} & y_{k_1+1} \ldots y_\ell f_1(g, x, y_1, \ldots, y_{k_1}) s_1(y_1, \ldots, y_{k_1}) t(x) + \\ & y_{k_2+1} \ldots y_\ell f_2(g, x, y_1, \ldots, y_{k_2}) s_2(y_1, \ldots, y_{k_2}) t(x) + \ldots + \\ & y_{k_m+1} \ldots y_\ell f_m(g, x, y_1, \ldots, y_{k_m}) s_m(y_1, \ldots, y_{k_m}) t(x)\end{aligned} \tag{8}$$

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 total $O(2^\ell)$

Algorithm 3 $\mathbf{A}_{h_g} \leftarrow \mathsf{Initialize\_PhaseOne}(f_1, \ldots, f_m, s_1, \ldots, s_m, \mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_m}, g)$

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 $|\mathbf{A}_{s_1}| + \ldots + |\mathbf{A}_{s_m}| = 2^\ell$.
Output: Bookkeeping table $\mathbf{A}_{h_g}$;

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

nonzero elements. Besides, we require that $2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = 2^\ell$. It is not hard to see that Equation 7 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 $2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = O(S_i)$ in Equation 7.

We use the same intuition in [20] of dividing the sumcheck process into two phases, one is for x and the other is for y. We rewrite Equation 8 as follows:

$$\sum_{x \in \{0,1\}^\ell} t(x) h_g(x), \tag{9}$$

Where

$$\begin{aligned} h_g(x) = \sum_{y \in \{0,1\}^\ell} & y_{k_1+1} \ldots y_\ell f_1(g, x, y_1, \ldots, y_{k_1}) s_1(y_1, \ldots, y_{k_1}) + \\ & y_{k_2+1} \ldots y_\ell f_2(g, x, y_1, \ldots, y_{k_2}) s_2(y_1, \ldots, y_{k_2}) + \ldots + \\ & y_{k_m+1} \ldots y_\ell f_m(g, x, y_1, \ldots, y_{k_m}) s_m(y_1, \ldots, y_{k_m})) \end{aligned} \tag{10}$$

Phase one. With the formula above, in the first ℓ rounds, the prover and the verifier are running exactly a sumcheck on a product of two multilinear functions $t \cdot h_g$, since functions t and h_g can be viewed as functions only in x, and y can be considered constant (it is always summed on the hypercube). To compute the sumcheck messages for the first ℓ rounds, given their bookkeeping tables, this will take $O(2^\ell)$ time in accordance with Lemma 1. We now 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 hypercube are known. Initializing the bookkeeping table for h_g in $O(2^\ell)$ time is more challenging but we can leverage the sparsity of f_i. And we have the lemma as follows:

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

The proof for Lemma 2 can be found in [20].

Lemma 3. *The bookkeeping table $\boldsymbol{A}_{h_g}$ 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 2, given the evaluations of $I(g,z)$ for all $z \in \{0,1\}^\ell$, the prover can iterate all $(z,y) \in \mathcal{N}_x$ for all x to compute $\mathbf{A}_{h_g}$. The full algorithm is presented in Algorithm 3. Since each s_i is a multilinear extension of an array and therefore the evaluations on the hypercube are known. So we use $\mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_m}$ as the input in Algorithm 3. $|\mathbf{A}_{s_1}| + \ldots + |\mathbf{A}_{s_m}| = 2^\ell$ as $2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = 2^\ell$.

Procedure $\mathsf{Precompute}(g)$ is to evaluate $\mathbf{G}[z] = I(g,z) = \prod_{i=1}^{\ell}((1-g_i)(1-z_i)+g_iz_i))$ for $z \in \{0,1\}^\ell$. By the closed-form of $I(g,z)$, 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 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, this 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 $\mathsf{SumCheckProduct}(h_g(x), \mathbf{A}_{h_g}, t(x), \mathbf{A}_t, u_1, \ldots, u_\ell)$ in Algorithm 2 and the total complexity for phase one is $O(2^\ell)$.

Phase two. At this point, all variables in x have been bounded to random numbers u. In the second phase, the equation to sum on becomes

$$t(u) \sum_{y\in\{0,1\}^\ell} (\sum_{i=1}^{m} y_{k_i+1} \ldots y_\ell f_i(g,u,y_1,\ldots,y_{k_i}) s_i(y_1,\ldots,y_{k_i}))$$

Note here that $t(u)$ is merely a constant which we already computed in phase one. For the part behind the summation symbol on y, it has l products of two multilinear functions to sum. If we naively apply Algorithm 2 to each product, the prover runs in $O(m \cdot 2^\ell)$ time instead of only $O(2^\ell)$. Fortunately, we observe that we can merge some products dynamically during the sumcheck process, which reduces the number of products and removes the m factor in the complexity. To achieve the linear prover time, we generalize Lemma 1 to Lemma 4 for the summation of multiple products of multilinear functions.

Lemma 4. *Suppose we have $2m$ multilinear functions $f_1, f_2, \ldots, f_m$ 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 $2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = 2^\ell$, given the* bookkeeping *tables $\boldsymbol{A}_{f_1}, \ldots, \boldsymbol{A}_{f_m}$ for $f_1, \ldots, f_m$ and $\boldsymbol{A}_{g_1}, \cdot, \boldsymbol{A}_{g_m}$ for $g_1, \ldots, g_m$, the prover in Protocol 1 for $\sum_{i=1}^{m} \sum_{y\in\{0,1\}^{k_i}} f_i(y) \cdot g_i(y) = \sum_{y\in\{0,1\}^\ell} \sum_{i=1}^{m} y_{k_i+1} \ldots y_\ell f_i(y_1,\ldots,y_{k_i}) \cdot g_i(y_1,\ldots,y_{k_i})$ runs in $O(2^\ell)$ time.*

Proof. We present Algorithm 4 for the prover in the sumcheck. $\mathcal{P}$ runs in $O(2^\ell)$ for step 1-3 as $|\mathbf{A}_{f_1}| + \ldots + |\mathbf{A}_{f_m}| = |\mathbf{A}_{g_1}| + \ldots + |\mathbf{A}_{g_m}| = 2^\ell$. $\mathcal{P}$ runs in $O(2^\ell)$ for step 5-12 as the total number of the operations is $O(2^{k_1} + 2^{k_2} + \ldots + 2^{k_m}) = O(2^\ell)$. So $\mathcal{P}$ runs in $O(2^\ell)$ time for Algorithm 4. □

The sumcheck polynomial for phase two has the same form in Lemma 4. To compute the sumcheck messages for the last ℓ rounds, given their bookkeeping tables, this will take $O(2^\ell)$ time by Lemma 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 $2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = O(2^\ell)$. 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 5. *Let $\mathcal{N}_y$ be the set of $(z,x) \in \{0,1\}^{2\ell}$ such that $f_i'(z,x,y) = y_{k_i+1} \ldots y_\ell\ f_i(z,x,y_1,\ldots,y_{k_i})$ is non-zero for some $1 \leq i \leq m$. Then for all $y \in \{0,1\}^\ell$, it is $f_i'(g,u,y) = \sum_{(z,x)\in\mathcal{N}_y} I(g,z)I(u,x)f'(z,x,y)$, where $I(g,z) = \prod_{i=1}^{\ell}((1-g_i)(1-z_i)+g_iz_i))$ and $I(u,x) = \prod_{i=1}^{\ell}((1-u_i)(1-x_i)+u_ix_i))$.*

Algorithm 4 $\{a_1, \ldots, a_\ell\} \leftarrow \mathsf{SumCheckProduct}(f_1, \mathbf{A}_{f_1}, g_1, \mathbf{A}_{g_1}, \ldots, f_m, \mathbf{A}_{f_m}, g_m, \mathbf{A}_{g_m}, r_1, \ldots, r_\ell)$

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 $|\mathbf{A}_{f_1}| + \ldots + |\mathbf{A}_{f_m}| = |\mathbf{A}_{g_1}| + \ldots + |\mathbf{A}_{g_m}| = 2^\ell$.

Output: ℓ sumcheck messages for $\sum_{y\in\{0,1\}^\ell} \sum_{i=1}^{m} y_{k_i+1} \ldots y_\ell f_i(y_1, \ldots, y_{k_i}) \cdot g_i(y_1, \ldots, y_{k_i})$. Each message a_i consists of 3 elements (a_{i0}, a_{i1}, a_{i2});

1: **for** $i = 1, \ldots, m$ **do**
2: $\quad \mathcal{F}_i \leftarrow \mathsf{FunctionEvaluations}(f_i, \mathbf{A}_{f_i}, r_1, \ldots, r_{k_i})$
3: $\quad \mathcal{G}_i \leftarrow \mathsf{FunctionEvaluations}(g_i, \mathbf{A}_{g_i}, r_1, \ldots, r_{k_i})$
4: $temp = 0$
5: **for** $i = 0, \ldots, m$ **do**
6: $\quad$ **if** $i > 0$ **then**
7: $\quad\quad temp = temp + f_i(r_1, \ldots, r_{k_i}) \cdot g_i(r_1, \ldots, r_{k_i})$
8: $\quad$ **for** $j = k_i + 1, \ldots, k_{i+1}$ **do** *// Suppose* $k_0 = 0 < k_1 \le \ldots \le k_m \le k_{m+1} = \ell$
9: $\quad\quad$ **if** $j \le \ell$ **then**
10: $\quad\quad\quad$ **for** $q \in \{0, 1, 2\}$ **do**
11: $\quad\quad\quad\quad a_{jq} = \sum_{t=i+1}^{m} \sum_{b\in\{0,1\}^{k_t - j}} f_t(r_1, \ldots, r_{j-1}, q, b) \cdot g_t(r_1, \ldots, r_{j-1}, q, b) + q \cdot temp$
// All evaluations needed are in $\mathcal{F}_i$ *and* $\mathcal{G}_i$.
12: $\quad\quad\quad\quad temp = temp \cdot r_j$
13: **return** $\{a_1, \ldots, a_\ell\}$;

Lemma 5 is a generalization of Lemma 2 and we omit the proof.

Lemma 6. *The bookkeeping table* $\boldsymbol{A}_{f_1}, \ldots, \boldsymbol{A}_{f_m}$ *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 2, given the evaluations of $I(g, z)$ and $I(u, x)$ 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 5.

$\mathcal{P}$ runs procedure $\mathsf{Precompute}(g)$ and $\mathsf{Precompute}(u)$ in $O(2^\ell)$ time as we have shown in the proof of Lemma 3. Step 4-5 computes $f_i(y_1, \ldots, y_{k_i})$ using Lemma 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 5. □

With the bookkeeping tables, the prover runs $\mathsf{SumCheckProduct}(f_1, \mathbf{A}_{f_1}, g_1, \mathbf{A}_{g_1}, \ldots, f_m, \mathbf{A}_{f_m}, g_m, \mathbf{A}_{g_m}, r_1, \ldots, r_\ell)$ in Algorithm 4 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 8.

Step one with linear prover time.. Finally, the sumcheck protocol for Equation 7 can be decomposed into several instances that have the form of Equation 8. The term

$$\begin{aligned} &\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{mult}_{i+1,i+1}(z, x, y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+1}(y)) + \ldots \\ &+ y_{s_{i,d}+1} \ldots y_{s_{i,i+1}} \tilde{mult}_{i+1,d}(z, x, y_1, y_2, \ldots, y_{s_{i,d}})(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,d}(y_1, y_2, \ldots, y_{s_{i,d}})) \end{aligned} \tag{11}$$

Algorithm 5 $\mathbf{A}_{f_1}, \ldots, \mathbf{A}_{f_m} \leftarrow \mathsf{Initialize\_PhaseTwo}(f_1, \ldots, f_m, g, u)$

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: $\mathbf{G} \leftarrow \mathsf{Precompute}(g)$
2: $\mathbf{U} \leftarrow \mathsf{Precompute}(u)$
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 $f_i(z, x, y_1, \ldots, y_{k_i})$ is non-zero **do**
5: $\quad \mathbf{A}_{f_i}[y_1, \ldots, y_{k_i}] = \mathbf{A}_{f_i}[y_1, \ldots, y_{k_i}] + \mathbf{G}[z] \cdot \mathbf{U}[x] \cdot f_i(z, x, y_1, \ldots, y_{k_i})$
6: **return** $\mathbf{A}_{f_1}, \mathbf{A}_{f_2}, \ldots, \mathbf{A}_{f_m}$;

is exactly the same as Equation 8. For the term of

$$\begin{aligned}\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} & (\tilde{add}_{i+1,i+1}(z,x,y)(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,i+1}(y))+\ldots \\ +y_{s_{i,d}+1}\ldots y_{s_{i,i+1}} & \tilde{add}_{i+1,d}(z,x,y_1,y_2,\ldots,y_{s_{i,d}})(\tilde{V}_{i,i+1}(x)+\tilde{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}})),\end{aligned} \tag{12}$$

it can be rewritten as

$$\begin{aligned}&\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{add}_{i+1,i+1}(z,x,y)(\tilde{V}_{i,i+1}(x)+\ldots+\tilde{V}_{i,d}(x)) \\ +&\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{add}_{i+1,i+1}(z,x,y)(\tilde{V}_{i,i+1}(y)+\ldots+\tilde{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}}))\end{aligned} \tag{13}$$

The first sum can be computed using the same protocol without $s_i(x)$, and the second sum can be computed without $t(x)$. 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, and sum up the individual messages and send them to the veriifer.

Verifier time and proof size for all sumcheck protocols on Equation 4. The verifier time for all sumcheck protocols on Equation 7 is the same as Protocol 2. $\mathcal{V}$ still runs in $O(d \log |C|)$ time to verify all sumcheck statements based on Equation 7. The proof size becomes $O(d \log |C| + d^2)$ as the proof contains extra $d - i$ values for each layer.

3.3 Combining Multiple Claims in Linear Time

By executing the sumcheck protocol on Equation 7, $\mathcal{P}$ and $\mathcal{V}$ reduce an evaluation of the multiliear extension of a layer to multiple evaluations of multilinear extensions defined by values in 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 7 in Section 3.2, now the verifier have 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 an approach that not only combines these evaluations on the subsets to a single evaluation of the multilinear extension of the entire layer i, but also incurs only a prover time linear to the size of the circuit. The key idea is that instead of trying to come up with a complicated protocol to do so, we simply model the computation as an arithmetic circuit! The circuit takes the values $\mathbf{V}_i$ of the entire

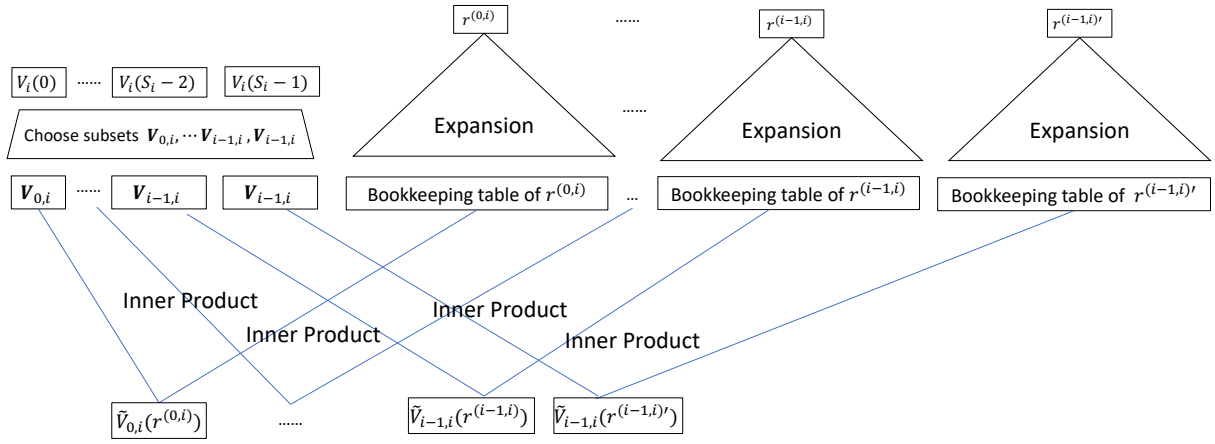


Figure 1: The struture of the circuit C_i with input $\mathbf{V}_i$ and $r^{(0,i)},\ldots,r^{(i-1,i)},r^{(i-1,i)'}$ to compute $\tilde{V}_{0,i}(r^{(0,i)}),\tilde{V}_{i-1,i}(r^{(i-1,i)}),\tilde{V}_{i-1,i}(r^{(i-1,i)'})$

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 1.

The output of the circuit is exactly the multiple evaluations of the multilinear extensions, which is 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 randomness used to compute the output. As the latter is known to $\mathcal{V}$, $\mathcal{V}$ can compute it locally. In this way, using the circuit, $\mathcal{P}$ and $\mathcal{V}$ reduce multiple claims about subsets of layer i to one claim about $\tilde{V}_i$.

Another tricky part is that the size of the circuit is not optimal if implemented naively. Note that for $\mathbf{V}_i$, the circuit only selects multiple subsets out of it, which can be done in one layer (left side of Figure 1). However, for the randomness, the circuit has to expand them to the bookkeeping tables in exactly the same way as described in Algorithm 1, which takes logarithmic layers. The circuit then computes the inner product between a subset and its corresponding bookkeeping table, which gives the evaluation of its multilinear extension. If we simply relay the subsets of $\mathbf{V}_i$ by logarithmic layers in the circuit, the size of the circuit is no longer linear. It becomes $O(S_i\log S_i)$ and introduces an logarithmic overhead to the whole protocol. Fortunately, we can use the method proposed in [22] to feed input to different layers. We input $\mathbf{V}_i$ at one layer before the inner product instead of from the top, as shown in Figure 1. Now the size of the circuit is linear to the total size of all the subsets. We give the formal protocol in Protocol 3.

Protocol 3. *Let $\mathbb{F}$ be a prime field. Let C_i be an arithmetic circuit with the input **in** consisting of two parts, i.e., $\boldsymbol{V}_i = (V_i(0), \ldots, V_i(S_i - 1))$ and $\boldsymbol{R} = (r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'})$, and the output **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 $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$. We know $|\boldsymbol{V}| = |\boldsymbol{R}|$ and we put $\boldsymbol{V}$ and $\boldsymbol{R}$ in the same layer and do inner product on pairs to output **out**. Without loss of generality, assume $|\boldsymbol{V}|$ and $|\boldsymbol{R}|$ are the power of 2 and we can pad them if not. The construction of C_i is shown in Figure 1.*

- *$\mathcal{P}$ and $\mathcal{V}$ invoke Protocol 2 on inner products to reduce the claim about **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 1, separately. For the left part, it reduces the claim about $\boldsymbol{V}(r)$ to the claim about $\boldsymbol{V}_i(r^{(i)})$ by 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 `1`; otherwise, output `0`.*

Efficiency. In order to analze the prover time for Subprotocol 3, we consider the specific structure of C_i, as shown in Figure 1. 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 1, 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 there are at most total $2|C|$ outwires in the circuit.

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 T_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 $O(\log^2 |C| + d \log |C| + T_i)$ and the proof size is $O(\log^2 |C|)$ by Theorem 1. In total for all layers, $\mathcal{V}$ runs in $O(d \log^2 |C| + d^2 \log |C| + T)$ time and the proof size is $O(d \log^2 |C|)$, where $T = T_1 + T_2 + \ldots + T_d$.

3.4 The Full Protocol for General Arithmetic Circuits

Combining the two steps together, we give the full protocol of the generalized GKR for arbitrary arithmetic circuits in Protocol 4.

Protocol 4. *Let $\mathbb{F}$ be a prime field. Let C: $\mathbb{F}^n \to \mathbb{F}^k$ be a d-depth unlayered arithmetic circuit. $\mathcal{P}$ wants to convince that $C(\boldsymbol{in}) = \boldsymbol{out}$ where $\boldsymbol{in}$ is the input from $\mathcal{V}$, and $\boldsymbol{out}$ is the output. Without loss of generality, assume n is the power 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*

$$\begin{aligned}\tilde{V}_0(g) = &\sum_{x,y\in\{0,1\}^{s_{0,1}}} (\tilde{add}_{1,1}(g,x,y)(\tilde{V}_{0,1}(x) + \tilde{V}_{0,1}(y)) + \dots \\ &+ y_{s_{0,d}+1}\dots y_{s_{0,1}} \tilde{add}_{1,d}(g,x,y_1,y_2,\dots,y_{s_{0,d}})(\tilde{V}_{0,1}(x) + \tilde{V}_{0,d}(y_1,y_2,\dots,y_{s_{0,d}}))) \\ &+ \sum_{x,y\in\{0,1\}^{s_{0,1}}} (\tilde{mult}_{1,1}(g,x,y)(\tilde{V}_{0,1}(x)\tilde{V}_{0,1}(y)) + \dots \\ &+ y_{s_{0,d}+1}\dots y_{s_{0,1}} \tilde{mult}_{1,d}(g,x,y_1,y_2,\dots,y_{s_{0,d}})(\tilde{V}_{0,1}(x)\tilde{V}_{i,d}(y_1,y_2,\dots,y_{s_{0,d}})))\end{aligned}$$

   At the end of the protocol, $\mathcal{V}$ receives $\tilde{V}_{0,1}(r^{(0,1)'}), \tilde{V}_{0,1}(r^{(0,1)}), \tilde{V}_{0,2}(r^{(0,2)}), \dots, \tilde{V}_{0,d}(r^{(0,d)})$. $\mathcal{V}$ 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 $\mathtt{0}$ and aborts.

3. *For $i = 1, ..., d-1, d$:*
   (a) *Given $\tilde{V}_{0,i}(r^{(0,i)}), \dots, \tilde{V}_{i-1,i}(r^{(i-1,i)'}), V_{i-1,i}(r^{(i-1,i)})$ and $r^{(0,i)}, r^{(1,i)}, \dots, r^{(i-1,i)'}, r^{(i-1,i)}$, $\mathcal{P}$ and $\mathcal{V}$ invoke Protocol 3 to build a layered arithmetic circuit C_i with input of $\boldsymbol{V}_i = V_i(0), \dots, V_i(S_i - 1)$ and $\boldsymbol{R} = (r^{(0,i)}, r^{(1,i)}, \dots, r^{(i-1,i)'}, r^{(i-1,i)})$. After running Protocol 3, if $\mathcal{V}$ in Protocol 3 does not abort, $\mathcal{V}$ receives $\tilde{V}_i(r^{(i)})$ for some randomness $r^{(i)} \in \mathbb{F}^{s_i}$.*
   (b) *If $i < d$, $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck on the equation*

$$\begin{aligned}\tilde{V}_i(r^{(i)}) = &\sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{add}_{i+1,i+1}(r^{(i)},x,y)(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,i+1}(y)) + \dots \\ &+ y_{s_{i,d}+1}\dots y_{s_{i,i+1}} \tilde{add}_{i+1,d}(r^{(i)},x,y_1,y_2,\dots,y_{s_{i,d}})(\tilde{V}_{i,i+1}(x) + \tilde{V}_{i,d}(y_1,y_2,\dots,y_{s_{i,d}}))) \\ &+ \sum_{x,y\in\{0,1\}^{s_{i,i+1}}} (\tilde{mult}_{i+1,i+1}(r^{(i)},x,y)(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,i+1}(y)) + \dots \\ &+ y_{s_{i,d}+1}\dots y_{s_{i,i+1}} \tilde{mult}_{i+1,d}(r^{(i)},x,y_1,y_2,\dots,y_{s_{i,d}})(\tilde{V}_{i,i+1}(x)\tilde{V}_{i,d}(y_1,y_2,\dots,y_{s_{i,d}})))\end{aligned}$$

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

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

Theorem 2. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a depth-d general arithmetic circuit. Protocol 4 is an interactive proof for the function computed by C with soundness $O(d^2 \log^2 |C|/|\mathbb{F}|)$. The running time of the prover $\mathcal{P}$ is $O(|C|)$. The proof size is $O(d \log^2 |C| + d^2)$. Let the time to evaluate all $\tilde{add}_{i,j}$ and $\tilde{mult}_{i,j}$ at random points be T', the time to evaluate all $\tilde{add}$ and $\tilde{mult}$ in C_i at random points be T_i, and $T = T_1 + T_2 + \ldots + T_d$, the running time of $\mathcal{V}$ is $O(n + d^2 \log |C| + d \log^2 |C| + T + T')$.*

Proof. **Completeness**. The completeness is straightforward by the completeness of the sumcheck protocol and the completeness of Protocol 2 in Theorem 1.

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 $\tilde{V}_j(r^{(j)}) = \tilde{V}'_j(r^{(j)})$ for $j > i$ but $\tilde{V}_i(r^{(i)}) \neq \tilde{V}'_i(r^{(i)})$, which event is defined as E_i. When running the sumcheck protocol for $\tilde{V}_i(r^{(i)})$, at the end of the sumcheck, $\mathcal{V}$ receives the messages of $\tilde{V}_{i,i+1}(r^{(i,i+1)'}), \tilde{V}_{i,i+1}(r^{(i,i+1)}), \ldots, \tilde{V}_{i,d}(r^{(i,d)})$ and check the last statement of the sumcheck protocol. There are two cases:

- Case 1: all these values are consistent with the correct values for circuit C with input **in**.
- Case 2: there exists some value $\tilde{V}_{i,l}(r^{(i,l)})$ or $\tilde{V}_{i,i+1}(r^{(i,i+1)'})$ such that $\tilde{V}_{i,l}(r^{(i,l)}) \neq \tilde{V}'_{(i,l)}(r^{(i,l)})$ or $\tilde{V}_{i,i+1}(r^{(i,i+1)'}) \neq \tilde{V}'_{i,i+1}(r^{(i,i+1)'})$. We define the event that $\tilde{V}_{i,l}(r^{(i,l)}) \neq \tilde{V}'_{(i,l)}(r^{(i,l)})$ as $E_{i,l}$. ($E_{i,i+1}$ represents $\tilde{V}_{i,i+1}(r^{(i,i+1)}) \neq \tilde{V}'_{(i,i+1)}(r^{(i,i+1)})$ or $\tilde{V}_{i,i+1}(r^{(i,i+1)'}) \neq \tilde{V}'_{i,i+1}(r^{(i,i+1)'})$)

The probability of case 1 is at most $\frac{2\lceil \log S_i \rceil}{|\mathbb{F}|}$ by the soundness of the sumcheck protocol. For case 2, that means there exists $l > i$ such that $C_i(\mathbf{in}_{C_i}) \neq \mathbf{out}_{C_i}$ or $\mathbf{in}_{C_i}$ is not consistent with $\mathcal{V}'_l(0), \ldots, \mathcal{V}'_l(S_l - 1)$.

If it is the latter, then $\tilde{V}_l(r^{(i+1)}) = \tilde{V}'_l(r^{(i+1)})$ with probability at most $\frac{\lceil \log S_i \rceil}{|\mathbb{F}|}$ by Schwartz-Zippel Lemma [13, 25].

If it is the former, from Section 3.3, we know the number of gates in C_l is at most $8 \sum_{k=0}^{S_l - 1} S_{k,l} \leq 8|C|$. The depth of C_i is at most $2\lceil \log S_l \rceil$ while the input size of the randomness is at most $d \log |C|$. Therefore, according to the soundness of Protocol 2, $C_i(\mathbf{in}_{C_i}) \neq \mathbf{out}_{C_i}$ but $\mathcal{V}$ in the subprotocol outputs 1 with probability at most $O(\frac{\log S_i \cdot \log |C|}{|\mathbb{F}|}) \leq O(\frac{\log^2 |C|}{|\mathbb{F}|})$.

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

$$
\begin{aligned}
\Pr[C(\mathbf{in}) \neq \mathbf{out} \wedge \mathcal{V} \text{ outputs } 1] &\leq \Pr[\exists i, E_i] \\
&\leq \Pr[E_0] + \Pr[E_1] + \ldots + \Pr[E_{d-1}] \\
&\leq \frac{2\lceil \log S_0 \rceil}{|\mathbb{F}|} + \Pr[\exists l > 0, E_{0,l}] + \frac{2\lceil \log S_1 \rceil}{|\mathbb{F}|} + \Pr[\exists l > 1, E_{1,l}] \\
&+ \ldots + \frac{2\lceil \log S_{d-1} \rceil}{|\mathbb{F}|} + \Pr[\exists l > d-1, E_{d-1,l}] \\
&\leq O(\frac{d \log |C|}{|\mathbb{F}|}) + \frac{d \log^2 |C|}{|\mathbb{F}|} + \frac{(d-1) \log^2 |C|}{|\mathbb{F}|} + \ldots + \frac{\log^2 |C|}{|\mathbb{F}|} \\
&\leq O(\frac{d^2 \log^2 |C|}{|\mathbb{F}|})
\end{aligned}
$$

Efficiency. Combining the efficiency analysis of Section 3.2 and Section 3.3, the prover runs in $O(|C|)$ time to generate the proof for Protocol 4. The total verifier time is $O(d^2 \log |C| + d \log^2 |C| + T + T')$ and the total proof size is $O(d \log |C| + d \log^2 |C| + d + d - 1 + \ldots + 1) = O(d \log^2 |C| + d^2)$. [2]

□

[2] Note here that the term d^2 in the proof size and verifier time is also bounded by $O(|C|)$ based on the same analysis

Removing the overhead on the proof size. As stated in Theorem 2, our new GKR protocol has a linear prover time for arbitrary arithmetic circuits. However, it does introduces an overhead of $O(\log |C|)$ on the proof size compared to the original GKR protocol. We can actually remove this overhead, with a compromise on the verifier time. Note that the term $O(d \log^2 |C|)$ on the proof size comes from Protocol 3 to combine multiple claims into one using the circuit in Figure 1. As the depth of the circuit is $O(\log |C|)$, the proof size is $O(\log^2 |C|)$ for one layer and $O(d \log^2 |C|)$ in total. However, we can actually reduce the depth of the circuit simply by asking the verifier to compute the expansions of the randomness on her own. In this way, the circuit directly takes them as input, computes their inner products with the subsets of $\mathbf{V}_i$, which can be done in constant depth (the summations in the inner products can be computed using a single sumcheck protocol in one layer [16]). In this way, the proof size can be reduced to $O(\log |C|)$, while the verifier time becomes linear to the total size of all the subsets, which is again bounded by the size of the circuit. In this way, we have a variant of the generalized GKR protocol with proof size $O(d \log |C| + d^2)$ ($O(d^2)$ from the sumcheck in the first step) and verifier time $O(|C|)$.

This improvement is not always a trade-off between the proof size and the verifier time. For a random circuit without any structure, the verifier time was already $O(|C|)$ in the worst case to evaluate all $\tilde{add}$ and $\tilde{mult}$ at random points. This approach reduces the proof size to $O(d \log |C| + d^2)$, which gives a generalized GKR protocol for arbitrary circuits with almost no overhead at all on the prover time, proof size and verifier time. The experimental results showed in Table 1 validate the conclusion (see Section 5). When the circuit is structured, it is possible to achieve sublinear verifier time (see below). In this case, the approach reduces the proof size by increasing the verifier back to linear.

Achieving sublinear verifier time. In the worst case for a random circuit, the verifier time is linear to the size of the circuit in both the original GKR protocol and our new protocol (T and T' in Theorem 1 and 2). Intuitively, this is the best to hope for schemes without setup, as the verifier has to know the function to verify and reading a random circuit already takes linear time. However, an important contribution of the GKR protocol is that the verifier time is sublinear for circuit with structures that can be described succinctly. Thus the protocol can be used to delegate computations to untrusted servers, as the verifier time is faster than computing the result locally.

Our new protocol inherits the feature of sublinear verifier time for classes of circuits appeared in the literature. For specific computations such as matrix multiplications, second frequency moment, image downscaling etc. [16, 18], they can already be expressed as layered circuits efficiently, and our generalized GKR protocol on these computations looks almost the same as the original GKR protocol, which achieves logarithmic verifier time. For data parallel circuits [16, 17], it is not hard to see that all the multilinear extensions $\tilde{add}, \tilde{mult}$ in both step 1 and for the circuit in Figure 1 in step 2 only depends on a single copy. Thus the verifier time is only linear to the size of a single copy, and is independent of the number of copies in the data parallel circuits. Finally, for log-space uniform circuits considered in the original GKR paper [9], there is a reduction to uniform log-space Turing machines in order to achieve poly-logarithmic verifier time. The reduction introduces a polynomial overhead on the prover time, and all the variants of the GKR protocol including ours can achieve poly-logarithmic verifier time using the reduction. Defining a large class of circuits that achieves both linear prover time and sublinear verifier is left as an interesting future work. We state a general theorem similar to prior work [18, 20, 21]:

Theorem 3. *If in addition $d = \mathsf{polylog}(|C|)$ and T and T' are both in $\mathsf{polylog}(|C|)$ time in Theorem 2, Protocol 4 is an interactive proof with succinct proof size and verifier time.*

4 Zero Knowledge Arguments Based on Generalized GKR

In this section, we build a new zero-knowledge argument protocol for general arithmetic circuits based on Protocol 4. The construction follows similar ideas proposed in [7, 20], but we introduce some optimizations

in Section 3.3 (i.e., d^2 comes from the input of circuit C_i, which cannot be larger than the size of C_i). Therefore, both the proof size and the verifier time are $O(|C|)$ in the worst case, which is the same as the original GKR protocol. This is also why there is no d^2 in the complexity of the prover time.

particularly for the second step of combining multiple points in order to preserve the efficiency of the prover. We introduce some definitions before presenting the formal protocol.

4.1 Definitions

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 3. *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 $(x, w) \in R$,*

$$\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}) \rangle(x) = 1$$

- ***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 $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $\pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp})$ and $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, x, \pi^*)$, the following probability is $\mathsf{negl}(\lambda)$:*

$$\Pr[(x; w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \mathsf{pp}) = 1]$$

- ***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*

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

*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 $\mathsf{poly}(\lambda, |x|, \log |w|)$.*

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 [2, 5, 6, 18, 20, 21].

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:

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

Definition 4. *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 following probability is negligible of* λ*:*

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

- ***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*

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

Zero knowledg GKR protocol. Following the idea of [7], Xie et al. [20] lifts the GKR protocol to be zero knowledge efficiently. First, the sumcheck protocol is replaced by a zero knowledge sumcheck protocol, which eliminates the leakage during the sumcheck protocol (See Section 4.2). Second, the multilinear extensions are replaced by *low degree extensions* such that evaluations on several points reveal no information about the underlying function. In particular, the prover selects a random polynomial $R_i(x_1, \ldots, x_{s_i}, z)$ and defines

$$\dot{V}_i(x_1, \ldots, x_{s_i}) \stackrel{def}{=} \tilde{V}_i(x_1, \ldots, x_{s_i}) + Z_i(x_1, \ldots, x_{s_i}) \sum_{z \in \{0,1\}} R_i(x_1, \ldots, x_{s_i}, z), \tag{14}$$

where $Z_i(x) = \prod_{i=1}^{s_i} x_i(1 - x_i)$ is the vanishing polynomial, i.e., $Z_i(x) = 0$ for all $x \in \{0,1\}^{s_i}$. $\dot{V}_i$ is the low degree extension of V_i as $\dot{V}_i(x) = \tilde{V}_i(x) = V_i(x)$ for all $x \in \{0,1\}^{s_i}$. As R_i is randomly selected by $\mathcal{P}$, revealing evaluations of $\dot{V}_i$ does not leak information about V_i and thus the values in the circuit. Meanwhile, the sumcheck equations in the GKR protocol still hold as the low degree extension also agrees with the function on the Boolean hypercube. The zero knowledge polynomial commitment scheme is also used to commit to these masking polynomials and later open them at random points. With these changes, we have the following lemma:

Lemma 7. *There exists a zero knowledge interactive proof protocol for layered arithmetic circuits* $C(\boldsymbol{in}) = \boldsymbol{out}$ *by replacing* $\tilde{V}_i$ *with* $\dot{V}_i$ *and the sumcheck with ZK sumcheck in Protocol 2. The protocol satisfies the completeness and the soundness in Definition 1. In addition, for every verifier* $\mathcal{V}^*$ *and every layered circuit* C, *there exists a simulator* $\mathcal{S}$ *such that given the oracle access to the output of* $\dot{V}_d(u^{(d)})$ *and* $\dot{V}_d(v^{(d)})$ *in step 4 of Protocol 2,* $\mathcal{S}$ *is able to simulate the view of* $\mathcal{V}^*$ *in step 1-3 of Protocol 2.*

By combining the zero knowledge GKR protocol with the zkPC scheme, Xie et al. [20] and Zhang et al. [21] construct zero knowledge arguments for layered arithmetic circuits.

4.2 Zero Knowledge Sumcheck

To build a zero knowledge argument for arbitrary arithmetic circuit using Protocol 4, we follow the same blueprint of [20] 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 7, 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 [20]. 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 $\mathsf{zkPC.Commit}$, and sends the verifier a claim $G = \sum\limits_{x_1,x_2,\ldots,x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell)$. The verifier picks a random number ρ, and execute a sumcheck protocol with the prover on

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

At the last round of this sumcheck, the prover opens the commitment of g at $g(r_1, \ldots, r_\ell)$ using $\mathsf{zkPC.Open}$, 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 4 ([20]). *Protocol 5 is complete and sound for the relationship of* $H = \sum\limits_{x_1,x_2,\ldots,x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell)$. *In addition, for every verifier* $\mathcal{V}^*$ *and every* ℓ*-variate polynomial* $f : \mathbb{F}^\ell \to \mathbb{F}$ *with variable degree* d*, there*

Protocol 5. *We assume the existence of a zkPC protocol defined in Section 4.1. For simplicity, we omit the randomness* r_f *and public parameters* pp, vp *without any ambiguity. To prove the claim* $H = \sum\limits_{x_1,x_2,\ldots,x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell)$*:*

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* $H = \sum\limits_{x_1,x_2,\ldots,x_\ell \in \{0,1\}} f(x_1, x_2, \ldots, x_\ell)$, $G = \sum\limits_{x_1,x_2,\ldots,x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell)$ *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* $\mathsf{zkPC.Open}$ *and* $\mathsf{zkPC.Verify}$*. 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* $f(r_1, \ldots, r_\ell)$*.*

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 7) of our new GKR protocol. It eliminates all the leakage during the sumcheck protocol.

4.3 Combining Multiple Claims to One with Zero Knowledge

Even with the zero knowledge sumcheck, the protocol still leaks information about values in the circuit. In particular, at the end of the zero knowledge sumcheck, $\mathcal{V}$ still needs an oracle access to $f(r_1, \ldots, r_\ell)$. When executed on Equation 7, the verifier evaluates all $\tilde{add}$ and $\tilde{mult}$ at the random point, and queries the prover for the evaluations of $\tilde{V}_{0,i}, \cdots, \tilde{V}_{i-1,i}$. These evaluations reveal information about values in the circuit.

To prevent this leakage, we use the same idea in [20] to replace them with their low-degree extensions $\dot{V}_{0,i}, \cdots, \dot{V}_{i-1,i}$. As explained above, Equation 7 still holds for low degree extensions, and $\mathcal{P}$ and $\mathcal{V}$ can execute the zero knowledge sumcheck on the new equation, which leaks no information about $V_{0,i}, \cdots, V_{i-1,i}$.

When going into the second step of combining multiple evaluations, now these low degree extensions have to be evaluated explicitly in a circuit similar to Figure 1. In order to do so, we let $\mathcal{P}$ input the masking polynomials used to define the low degree extensions, and let $\mathcal{V}$ input the vanishing polynomial as public inputs. $\mathcal{P}$ commits to the masking polynomials using the zkPC, and $\mathcal{P}$ and $\mathcal{V}$ execute the zero knowledge GKR to reduce the evaluations of the low degree extensions of the subsets to a single evaluation of the low degree extension of V_i.

To maintain the optimal prover time, we observe that the masking polynomials used in the low degree extensions can be simplified significantly. For $\dot{V}_{0,i}, \cdots, \dot{V}_{i-1,i}$, only a single evaluation of each polynomial is sent to the verifier. Hence it suffices to mask each $\tilde{V}_{j,i}$ with a single random number:

$$\dot{V}_{j,i}(z) \stackrel{def}{=} \tilde{V}_{j,i}(z) + Z_{j,i}(z) \cdot p_{j,i}, \tag{15}$$

where $Z_{j,i}(z) = \prod_{k=1}^{s_{j,i}} z_k(1 - z_k)$ and $p_{j,i} \in \mathbb{F}$ is randomly generated by $\mathcal{P}$. To be precise, $\mathcal{V}$ receives two evaluations $\dot{V}_{i-1,i}(r^{(i-1,i)})$ and $\dot{V}_{i-1,i}(r^{(i-1,i)'})$ for $\dot{V}_{i-1,i}$. Thus for $j = i-1$, $\mathcal{P}$ generates two random numbers of $p_{i-1,i}$ and $p'_{i-1,i}$ to define $\dot{V}_{i-1,i}$ and $\dot{V}'_{i-1,i}$ twice separately. Note that the sum on variable z in Equation 14 is not necessary, as these low degree extensions will not be used in the next round of the sumcheck. For $\dot{V}_i$, after combining the evaluations of the subsets, again the prover only sends a single evaluation of $\dot{V}_i$ to the verifier. Thus we define $\dot{V}_i(x_1, \ldots, x_{s_i}) = \tilde{V}_i(x_1, \ldots, x_{s_i}) + Z_i(x_1, \ldots, x_{s_i}) \sum_{z \in \{0,1\}} R_i(x_1, z)$, i.e., the masking polynomial in Equation 14 is set to be $R_i(x_1, z)$, a random multilinear polynomial with only two variables. With these low degree extensions, Equation 7 becomes

$$\begin{aligned}
\dot{V}_i(g) = \sum_{x,y \in \{0,1\}^{s_{i,i+1}}, w \in \{0,1\}} & [I(w,0)(\tilde{add}_{i+1,i+1}(g,x,y)(\dot{V}'_{i,i+1}(x) + \dot{V}_{i,i+1}(y_1, \ldots, y_{s_{i,i+1}})) \\
& + y_{s_{i,i+2}+1} \cdot \ldots \cdot y_{s_{i,i+1}} \tilde{add}_{i+1,i+2}(g,x,y_1,y_2,\ldots,y_{s_{i,i+2}})(\dot{V}'_{i,i+1}(x) + \dot{V}_{i,i+2}(y_1,y_2,\ldots,y_{s_{i,i+2}})) + \ldots \\
& + y_{s_{i,d}+1} \cdot \ldots \cdot y_{s_{i,i+1}} \tilde{add}_{i+1,d}(g,x,y_1,y_2,\ldots,y_{s_{i,d}})(\dot{V}'_{i,i+1}(x) + \dot{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}})) \\
& + \tilde{mult}_{i+1,i+1}(g,x,y)(\dot{V}'_{i,i+1}(x)\dot{V}_{i,i+1}(y_1,\ldots,y_{s_{i,i+1}})) \\
& + y_{s_{i,i+2}+1} \cdot \ldots \cdot y_{s_{i,i+1}} \tilde{mult}_{i+1,i+2}(g,x,y_1,y_2,\ldots,y_{s_{i,i+2}})(\dot{V}'_{i,i+1}(x)\dot{V}_{i,i+2}(y_1,y_2,\ldots,y_{s_{i,i+2}})) + \ldots \\
& + y_{s_{i,d}+1} \cdot \ldots \cdot y_{s_{i,i+1}} \tilde{mult}_{i+1,d}(g,x,y_1,y_2,\ldots,y_{s_{i,d}})(\dot{V}'_{i,i+1}(x)\dot{V}_{i,d}(y_1,y_2,\ldots,y_{s_{i,d}})) \\
& + I((x,y),\vec{0})Z_i(g)R_i(g_1,w)]
\end{aligned} \tag{16}$$

where $I(\vec{a}, \vec{b})$ is the identity polynomial $I(\vec{a}, \vec{b}) = 0$ iff $\vec{a} = \vec{b}$. The equation holds because $\dot{V}_{j,i}$ agrees with $\tilde{V}_{j,i}$ on the Boolean hypercube $\{0,1\}^{s_{i,j}}$, as $Z_{j,i}(z) = 0$ for binary inputs.

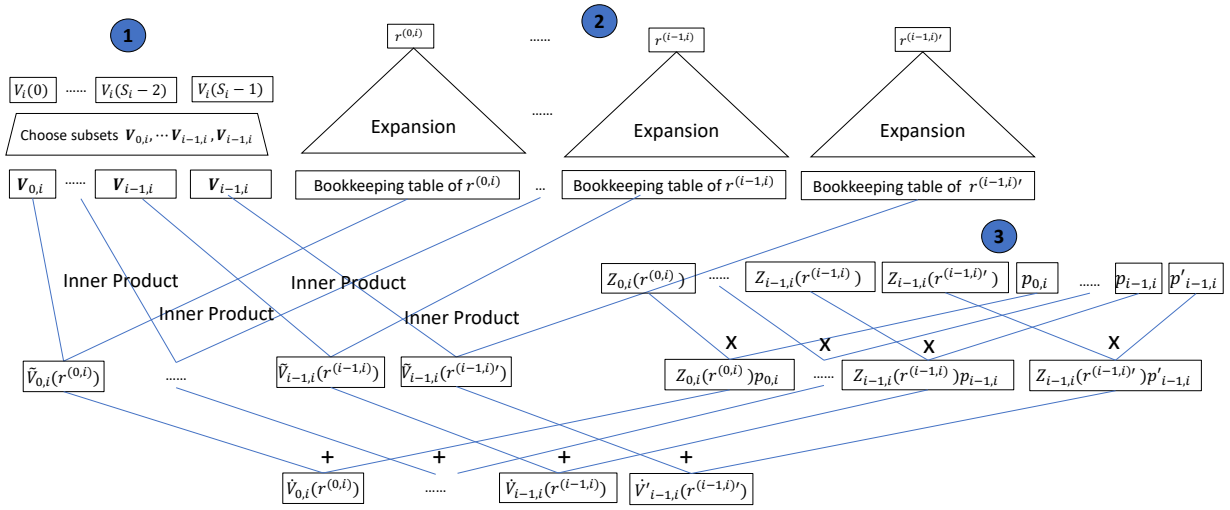


Figure 2: The struture of the circuit C'_i with input $\mathbf{V}_i$, $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$ and $p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i}$ to compute $\dot{V}_{0,i}(r^{(0,i)}), \dot{V}_{i-1,i}(r^{(i-1,i)}), \dot{V}'_{i-1,i}(r^{(i-1,i)'})$

Now $\mathcal{P}$ and $\mathcal{V}$ instead execute the zero-knowledge sumcheck protocol on Equation 16. 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 $Z_i(g), I(c,0), I((r^{(i,i+1)'}, r^{(i,i+1)}), \vec{0})$ 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.

New circuit to combine multiple evaluations in zero knowledge. To make step 3(a) of Protocol 4 work, we need to modify the circuit C_i in Figure 1 to reduce claims about $\dot{V}_{0,i}(r^{(0,i)}), \ldots, \dot{V}_{i-1,i}(r^{(i-1,i)}), \dot{V}'_{i-1,i}(r^{(i-1,i)'})$ to $\dot{V}_i(r^{(i)})$ for some randomness $r^{(i)}$ chose by the verifier. The new circuit C'_i is shown in Figure 2. As explained above, to compute $\dot{V}_{0,i}(r^{(0,i)'}), \ldots, \dot{V}_{i-1,i}(r^{(i-1,i)}), \dot{V}'_{i-1,i}(r^{(i-1,i)'})$, the circuit further takes as input $Z_{0,i}(r^{(0,i)}), \ldots, Z_{i-1,i}(r^{(i-1,i)}), Z_{i-1,i}(r^{(i-1,i)'})$ from the verifier and $p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i}$ from the prover. They are multiplied pairwisely and added to $\tilde{V}_{j,i}(r^{(j,i)})$, which gives exactly $\dot{V}_{j,i}(r^{(j,i)})$ by definition.

To execute the second step, the prover commits to the low degree extension of $p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i}$ using zkPC. $\mathcal{P}$ and $\mathcal{V}$ then run the zero knowledge GKR protocol on the circuit C'_i. By lemma 7, at the end of the protocol the verifier receives the evaluation of the low degree extension of the input at a random point $r^{(i)}$. The verifier then opens the low degree extension of $p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i}$ at the random point using the zkPC, shaves it off to obtain the evaluation of $\dot{V}_i(r^{(i)})$. The entire protocol is precisely the zero knowledge argument scheme in [20], where $p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i}$ are the witness from the prover, and V_i and $Z_{0,i}(r^{(0,i)}), \ldots, Z_{i-1,i}(r^{(i-1,i)}), Z_{i-1,i}(r^{(i-1,i)'})$ are the public inputs from the verifier. Similar to the version without zero knowledge in Section 3.3, the input can also come in different layers as shown in Figure 2 to achieve better efficiency. The new subprotocol with zero knowledge is presented in Protocol 6.

Protocol 6. *Let $\mathbb{F}$ be a prime field. Let C_i' be an arithmetic circuit with the input **in** consisting of four parts, i.e., $\boldsymbol{V}_i = (V_i(0), \ldots, V_i(S_i - 1))$, $\boldsymbol{R} = (r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'})$, $\boldsymbol{Z} = (Z_{0,i}(r^{(0,i)}), \ldots, Z_{i-1,i}(r^{(i-1,i)}), Z_{i-1,i}(r^{(i-1,i)'})$, $\boldsymbol{P} = (p_{0,i}, \ldots, p_{i-1,i}, p'_{i-1,i})$, and the output **out** $= (\dot{V}_{0,i}(r^{(0,i)}), \ldots, \dot{V}_{i-1,i}(r^{(i-1,i)}), \dot{V}'_{i-1,i}(r^{(i-1,i)'}))$. 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 $r^{(0,i)}, \ldots, r^{(i-1,i)}, r^{(i-1,i)'}$. Let $\tilde{\boldsymbol{V}}_i = (\tilde{V}_{0,i}(r^{(0,i)}), \ldots, \tilde{V}_{i-1,i}(r^{(i-1,i)}), \tilde{V}_{i-1,i}(r^{(i-1,i)'}))$ and $\boldsymbol{ZP} = (Z_{0,i}(r^{(0,i)})p_{0,i}, \ldots, Z_{i-1,i}(r^{(i-1,i)})p_{i-1,i}, Z_{i-1,i}(r^{(i-1,i)'})p'_{i-1,i})$. We use $R_{\boldsymbol{ZP}}$, $R_{\tilde{\boldsymbol{V}}_i}$, $R_{\boldsymbol{V}}$, $R_{\boldsymbol{T}}$, $R_{\boldsymbol{Z}}$, $R_{\boldsymbol{P}}$, $R_{\boldsymbol{R}}$ to mask multilinear extensions of $\boldsymbol{ZP}$, $\tilde{\boldsymbol{V}}_i$, $\boldsymbol{V}$, $\boldsymbol{T}_R$, $\boldsymbol{Z}$, $\boldsymbol{P}$, $\boldsymbol{R}$, respectively. We know $|\boldsymbol{V}| = |\boldsymbol{R}|$ and we put $\boldsymbol{V}$ and $\boldsymbol{R}$ in the same layer and do inner product on pairs to compute $\tilde{\boldsymbol{V}}_i$. We also put $\boldsymbol{Z}$ and $\boldsymbol{P}$ in the same layer, $\tilde{\boldsymbol{V}}_i$ and $\boldsymbol{ZP}$ in the same layer. Without loss of generality, assume $|\boldsymbol{V}|$, $|\boldsymbol{R}|$, $|\tilde{\boldsymbol{V}}_i|$ and $|\boldsymbol{P}|$are the power of 2 and we can pad them if not. The construction of C_i' is shown in Figure 2.*

- $\mathcal{G}(1^\lambda)$*: set* pp *as* $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$.
- $\mathcal{P}$ *commits* $\boldsymbol{P}(v) + \sum_{w\in\{0,1\}} R_{\boldsymbol{P}}(v_1, w)$, $R_{\boldsymbol{ZP}}$, $R_{\tilde{\boldsymbol{V}}_i}$, $R_{\boldsymbol{V}}$, $R_{\boldsymbol{T}}$, $R_{\boldsymbol{Z}}$, $R_{\boldsymbol{R}}$ *to* $\mathcal{V}$ *by* zkPC.Commit.
- $\mathcal{P}$ *and* $\mathcal{V}$ *invoke the zero knowledge GKR on the output layer to reduce the claim about* **out** *to the claim about* $\tilde{\boldsymbol{V}}_i$ *and the claim about* $\boldsymbol{ZP}$*:* $\tilde{\boldsymbol{V}}_i(t) + \sum_{w\in\{0,1\}} R_{\tilde{\boldsymbol{V}}_i}(t_1, w)$ *and* $\boldsymbol{ZP}(t) + \sum_{w\in\{0,1\}} R_{\boldsymbol{ZP}}(t_1, w)$.
- $\mathcal{P}$ *and* $\mathcal{V}$ *invoke the zero knowledge GKR on inner products to reduce the claim about* $\tilde{\boldsymbol{V}}_i(t) + \sum_{w\in\{0,1\}} R_{\tilde{\boldsymbol{V}}_i}(t_1, w)$ *to the claim about the layer of* $\boldsymbol{V}$ *and* $\boldsymbol{T}_R$*:* $\boldsymbol{V}(u) + \sum_{w\in\{0,1\}} R_{\boldsymbol{V}}(u_1, w)$ *and* $\boldsymbol{T}_R(u) + \sum_{w\in\{0,1\}} R_{\boldsymbol{T}}(u_1, w)$.
- $\mathcal{P}$ *and* $\mathcal{V}$ *invoke the zero knowledge GKR on the first part* ①*, the second part* ② *and the third part* ③ *of* C_i' *separately, as shown in Figure 2. For the first part, it reduces the claim about* $\boldsymbol{V}(u) + \sum_{w\in\{0,1\}} R_{\boldsymbol{V}}(u_1, w)$ *to the claim about* $\dot{V}_i(r^{(i)})$ *by one layer. For the second part, it reduces the claim about* $\boldsymbol{T}_R(u) + \sum_{w\in\{0,1\}} R_{\boldsymbol{T}}(u_1, w)$ *to the claim about* $\boldsymbol{R}(r^{(i)}) + \sum_{w\in\{0,1\}} R_{\boldsymbol{R}}(r_1^{(i)}, w)$*. For the third part, it reduces the claim about* $\boldsymbol{ZP}(t) + \sum_{w\in\{0,1\}} R_{\boldsymbol{ZP}}(t_1, w)$ *to the claim about the layer of* $\boldsymbol{Z}$ *and* $\boldsymbol{P}$*:* $\boldsymbol{Z}(u) + \sum_{w\in\{0,1\}} R_Z(v_1, w)$ *and* $\boldsymbol{P}(v) + \sum_{w\in\{0,1\}} R_{\boldsymbol{P}}(v_1, w)$.
- $\mathcal{V}$ *asks* $\mathcal{P}$ *to send* $\dot{V}_i(r^{(i)})$ *and checks the reduction for the first part.* $\mathcal{V}$ *computes* $\boldsymbol{R}(r^{(i)})$ *itself and uses* zkPC.Verify *to verify* $\sum_{w\in\{0,1\}} R_{\boldsymbol{R}}(r_1^{(i)}, w)$ *and checks the reduction for the second part.* $\mathcal{V}$ *computes* $\boldsymbol{Z}(u)$ *itself and uses* zkPC.Verify *to verify* $\sum_{w\in\{0,1\}} R_{\boldsymbol{Z}}(u_1, w)$ *and checks the reduction for the third part.* $\mathcal{V}$ *also checks the claim about* $\boldsymbol{P}(v) + \sum_{w\in\{0,1\}} R_{\boldsymbol{P}}(v_1, w)$ *by using* zkPC.Verify *for the third part. If all checks pass, output* 1*; otherwise, output* 0.

Theorem 5. *Protocol 6 is a zero-knowledge version of Protocol 3. 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 Protocol 6.*

Proof sketch. The completeness and the soundness inherits from the zero knowledge GKR and zkPC. For zero-knowledgeness, we combine the simulator $\mathcal{S}_1$ in zkPC and the simulator $\mathcal{S}_2$ in the zero knowledge GKR 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 $\tilde{V}_i$ according to the zero knowledge GKR.

Efficiency. Compared to Protocol 3, we add at most extra $O(d)$ private input of $\mathbf{P}$ in C_i'. When invoking the zero knowledge GKR, the verifier time and the proof size retain the same complexity as in Protocol 3. The prover time for C_i' becomes $O(\sum_{k=0}^{i-1} S_{k,i} + d \log d)$ using the zkPC scheme in [21], where zkPC.Commit and zkPC.Open cost prover time of $O(d \log d)$. Therefore, the total prover time for $C_1, \cdots, C_d$ becomes $O(|C| + d^2 \log d)$.Similar to the efficiency analysis of Theorem 2, the factor of d^2 is bounded by $O(|C|)$. Consequently, the total prover time for $C_1, \cdots, C_d$ is bounded by $|C| \log d$. For circuits with $d = O(\mathsf{polylog}|C|)$, the prover time remains $O(|C|)$.

4.4 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. We have the following theorem:

Theorem 6. *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 3. Moreover, for every $(C, x; w) \in \mathcal{R}$, the running time of $\mathcal{P}$ is $O(|C| + d^2 \log d + n \log n)$. The running time of $\mathcal{V}$ is $O(|x| + d \cdot \log^2 |C| + d^2 + T'')$, where T'' is the total time to compute all functions of $\tilde{add}$ and $\tilde{mult}$. The total proof size is $O(d \log^2 |C|)$. In case d is $\mathsf{polylog}(|C|)$ and T'' is also $\mathsf{polylog}(|C|)$, the protocol is a succinct argument.

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

To prove zero knowledge, consider a simulator $\mathcal{S}$ that calls the simulator $\mathcal{S}_1$ of zero knowledge sumcheck given in Theorem 4 for step 1, the simulator $\mathcal{S}_2$ of combining multiply claims to one claim with zero knowledge given in Theorem 5 for step 2 and the simulator $\mathcal{S}_3$ of zkPC in Definition 4 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 efficiency of Protocol 6, the efficency of Protocl 5 and the extra complexity of applying zkPC.Commit to the input layer demonstrated as before.

5 Implementation and Evaluation

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 1300 lines of code for the sumcheck protocol (Step 3(b) of Protocol 4) and 1600 lines for the protocol combining multiple evaluations into one (Protocol 3). We implement two variants of our interactive proofs as described in the optimization in Section 3.3. One is expanding the randomness in the circuit as shown in Figure 1 and the other is letting the verifier expand the randomness on her own. 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 [21], and we choose it so that our interactive proof protocols can be compatible with the polynomial commitments in [21] 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 of the experiments on our laptop with an Intel Core i5-7200 CPU with 2.50GHz and 8GB of RAM. Our current implementation is not parallelized and we only utilize a single CPU core in the experiments. We report the average running time of 10 executions.

5.1 Optimizations

We developed a concrete optimization during the implementation to reduce the size of the circuit in Figure 1. Though asymptotically the prover time for both the sumcheck and the combination of multiple points are linear, the majority of the running time in practice is on the latter. As we can see in Figure 1, the circuit C_i contains all subsets of $\mathbf{V}_{0,i}, \ldots, \mathbf{V}_{i-1,i}$ and one more $\mathbf{V}_{i-1,i}$ coming from layer $i-1$. The extra $\mathbf{V}_{i-1,i}$ represents the set of left input wires of layer $i-1$, which roughly equals to $\mathbf{V}_i$ as almost all gates in layer i connect to layer $i-1$. It contributes to almost half of the size of C_i. Instead, we replace the $\tilde{V}_{i-1,i}(x)$ in Equation 7 for layer $i-1$ by $\tilde{V}_i(x)$, which does not increase the size of the polynomial by much. We then

		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	0.225	0.848	3.254	0.096	0.373	1.429	83	95	107
	Our Scheme 1	0.116	0.389	1.244	0.040	0.130	0.432	276	396	534
	Our Scheme 2	0.052	0.188	0.670	0.024	0.080	0.307	105	124	143
$d = 75$	GKR	0.460	1.780	6.864	0.180	0.707	2.751	129	147	166
	Our Scheme 1	0.187	0.619	1.932	0.070	0.224	0.760	414	594	802
	Our Scheme 2	0.081	0.288	1.044	0.042	0.145	0.560	171	200	229

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

exclude the evaluation of this $\tilde{V}_i$ from the circuit, and instead combine it with the evaluation of $\tilde{V}_i$ from the input of C_i using the same random linear combination approach in the original GKR protocol. With this technique, we improve the overall prover time by $2\times$.

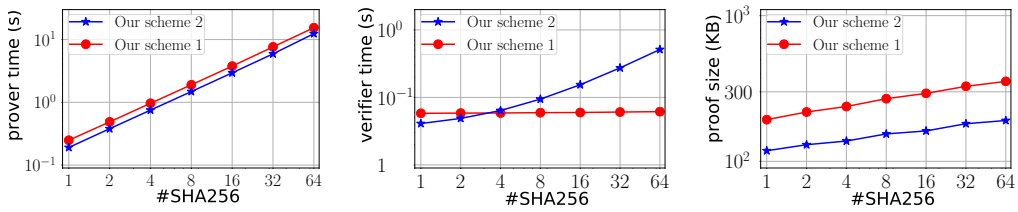
In particular, in our implementation, each gate in the original GKR protocol for layered circuits costs around 20 field multiplications in the prover time. With this optimization, in our protocol, each gate in the general circuit costs around 100 field multiplications. In the second variant without the expansion of the random points in the circuit, the constant is further reduced to 47. It demonstrates that our new protocols for general circuits not only preserves the linear prover time asymptotically, but also introduces a small overhead in practice. Namely, as long as the layered circuit is 2.4–5$\times$ larger than the corresponding general circuit, our new scheme will be faster on prover time.

5.2 Comparing to the GKR Protocol for Layered Circuits

In this section, we compare the performance of our new protocols with the original GKR protocol on general arithmetic circuits. For a fair comparison, we re-implement the GKR protocol for layered arithmetic circuits with the same field and the same libraries in C++. We generate random 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 with expansions in the circuit in Protocol 4 as scheme 1 and the one without the expansions discussed in Section 3.3 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 of these three schemes in Table 1.

First, when we transform the general circuits to layered circuits, the size of the circuit increases by 13$\times$ for $d = 50$ and by 19$\times$ 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 2-3$\times$ than the original GKR protocol, while our scheme 2 is faster by 4-5$\times$. When the depth is 75, the speedup increases to 3-4$\times$ for scheme 1 and 6-8$\times$ for scheme 2. The improvements match our analysis above on the blowup of the circuit size, and the concrete efficiency of our protocols and the GKR protocol. Finally, the prover time in all schemes grow linearly with the size of the circuit, and is very efficiency in practice.

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$\times$ larger than the GKR protocol, matching the $\log |C|$ overhead in the complexity of the proof size. However, with our optimization, the proof size of our second scheme is very close to the GKR protocol. It is only 1.3-1.4$\times$ larger, proving that this variant reduces the proof size significantly. 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.



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

Figure 3: Performance of our zero knowledge argument schemes.

As the circuits are generated randomly, the verifier time in all schemes is linear to the circuit size. Therefore, the comparisons on the verifier time of the three protocols are similar to the comparisons of the prover time. As shown in Table 1, our scheme 1 is faster by 2-3× than the GKR protocol on circuits with $d = 50$, and 3-4× faster for $d = 75$. Surprisingly, the verifier time of our scheme 2 is even faster than scheme 1. This is because both schemes have linear verifier time for random circuits. The time is actually dominated by the verifier time of the expansion circuit in Figure 1. By removing the expansion circuit, the verifier time in scheme 2 is faster than scheme 1, even though the verifier has to expand all the random points on her own. Therefore, we observe in the experiments that our scheme 2 improves the performance of scheme 1 on all the aspects on random circuits. Compared to the original GKR protocol, it is much faster on the prover time and the verifier time, and incurs only a small overhead on the proof size.

5.3 Experiments on Our Zero Knowledge Argument

In this section, we present the performance of our new zero knowledge argument for general arithmetic circuits, as described in Section 4. We use the zero knowledge polynomial commitment scheme in [21] to lift our new interactive proofs to zero knowledge arguments. As demonstrated in [20], the zero knowledge versions of the sumcheck protocol and the GKR protocol only incur a small overhead on the prover time, proof size and the verifier time. So we estimate the performance of our zero knowledge arguments by running the plain version of our interactive proof protocols, combined with the zkPC on the witness.

We do experiments on the benchmark of computing the hash functions of SHA-256. We modify the circuit generation file of [1, 20] to obtain the general arithmetic circuit for SHA-256. The circuit contains other types of gates such as subtraction, bit decomposition and reconstruction. We modify our protocols to support all these types of gates. Each SHA-256 circuit has 99,949 gates in total (around 2^{17}), with the input size of 7,226 (around 2^{13}). We vary the number of SHA-256 in the circuit from 1 to 64 and report the performance of our zero knowledge arguments built on scheme 1 and scheme 2 in Figure 3.

As shown in Figure 3, our schemes achieve good efficiency in practice. For the largest instance of 64 hashes, scheme 1 takes 15 seconds to generate the proof, the verifier time is 0.062 seconds and the proof size is 353KB. Scheme 2 takes 12 seconds to generate the proof, the verifier time is 0.5 seconds and the proof size is 191KB.

The prover time in both schemes increases roughly linearly as the number of hash functions increases, which validates our linear algorithm for the prover on the total size of the circuit. It does increase with the size of the input in $O(n \log n)$ time due to the zkPC scheme. Scheme 2 is better than scheme 1 in prover time and proof size because of the interactive proof part, as we already showed in the previous section. However, this time the verifier time of scheme 2 increases linearly as the number of hash functions, while the verifier time of scheme 1 stays almost the same. This is due to the sublinear verifier time of scheme 2 for structured (data parallel) circuits inherited from the original GKR protocol. All the multilinear extensions $\tilde{add}$ and $\tilde{mult}$ can be evaluated based on the wiring pattern of a single copy of SHA-256. For the proof size, both schemes increase sublinearly as our zero knowledge arguments are succinct as stated in Theorem 6. Scheme 2 has shorter proof than scheme 1 because of the optimization in Section 3.3. It gives a trade-off between the proof size and the verifier time.

Due to the space limit, we omit the experimental comparisons to other existing zero knowledge argument systems. We refer the readers to [18, 20, 21] for their performance on the same benchmark of SHA-256.

6 Conclusion

In this paper, we generalize the doubly efficient interactive proofs for layered arithmetic circuits to arbitrary arithmetic circuits. The new protocols preserves the optimal prover complexity that is linear to the size of the circuit. Experiments demonstrate that the new protocols can be implemented efficiently in practice, and their prover time is an order of magnitude faster than the original GKR protocol on the corresponding layered circuits.

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 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 DARPA.

References

[1] Libra implementation. https://github.com/sunblaze-ucb/fastZKP/tree/Libra

[2] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2017)

[3] Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Computational complexity **1**(1), 3–40 (1991)

[4] Baum, C., Bootle, J., Cerulli, A., Del Pino, R., Groth, J., Lyubashevsky, V.: Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In: Annual International Cryptology Conference. pp. 669–699. Springer (2018)

[5] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 103–128. Springer (2019)

[6] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: Proceedings of the Symposium on Security and Privacy (SP), 2018. vol. 00, pp. 319–338

[7] Chiesa, A., Forbes, M.A., Spooner, N.: A Zero Knowledge Sumcheck and its Applications. CoRR **abs/1704.02086** (2017), http://arxiv.org/abs/1704.02086

[8] Cormode, G., Mitzenmacher, M., Thaler, J.: Practical Verified Computation with Streaming Interactive Proofs. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. ITCS '12

[9] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating Computation: Interactive Proofs for Muggles. J. ACM **62**(4), 27:1–27:64 (Sep 2015)

[10] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)

[11] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic Methods for Interactive Proof Systems. J. ACM **39**(4), 859–868 (Oct 1992)

[12] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P 2013. pp. 238–252 (2013)

[13] Schwartz, J.T.: Probabilistic algorithms for verification of polynomial identities. In: International Symposium on Symbolic and Algebraic Manipulation. pp. 200–215. Springer (1979)

[14] Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)

[15] Shamir, A.: Ip= pspace. Journal of the ACM (JACM) **39**(4), 869–877 (1992)

[16] Thaler, J.: Time-Optimal Interactive Proofs for Circuit Evaluation. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013 (2013)

[17] Wahby, R.S., Ji, Y., Blumberg, A.J., Shelat, A., Thaler, J., Walfish, M., Wies, T.: Full accounting for verifiable outsourcing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM (2017)

[18] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943. IEEE (2018)

[19] Walfish, M., Blumberg, A.J.: Verifying computations without reexecuting them. Commun. ACM **58**(2), 74–84 (2015)

[20] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Annual International Cryptology Conference. pp. 733–764. Springer (2019)

[21] Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: S&P 2020

[22] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: Security and Privacy (SP), 2017 IEEE Symposium on. pp. 863–880. IEEE (2017)

[23] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A Zero-Knowledge version of vSQL. Cryptology ePrint (2017)

[24] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster verifiable RAM with program-independent preprocessing. In: Proceeding of IEEE Symposium on Security and Privacy (S&P) (2018)

[25] Zippel, R.: Probabilistic algorithms for sparse polynomials. In: International Symposium on Symbolic and Algebraic Manipulation. pp. 216–226. Springer (1979)