Almost optimal succinct arguments for Boolean circuit on RAM

Tiancheng Xie[1] and Tianyi Liu[2,3]

[1] Polyhedra Network
[2] University of Illinois Urbana-Champaign
[3] work was done on early 2023

Abstract. The significance of succinct zero-knowledge proofs has increased considerably in recent times. However, one of the major challenges that hinder the prover's efficiency is when dealing with Boolean circuits. In particular, the conversion of each bit into a finite field element incurs a blow-up of more than 100x in terms of both memory usage and computation time.

This work focuses on data-parallel Boolean circuits that contain numerous identical sub-circuits. These circuits are widely used in real-world applications, such as proving a large number of hash-preimages in zkEVM and zkBridge [1, 30]. We develop a method for constructing succinct arguments with $2^{-\lambda}$ soundness error and $O(\omega(1)\frac{N}{\log N}\log\log N)$ RAM operations, or $O(\frac{N}{\log N}\log\log N)$ finite field additions, along with a negligible number of finite field multiplications.

Our approach is based on using the GKR protocol [12] to obtain the succinct argument.

Keywords: Boolean circuits · zero-knowledge proofs · GKR

1 Introduction

Succinct arguments are proof systems that enable a prover to convince a verifier that a computation is correct using a proof of sub-linear size. These proof systems have recently garnered significant attention and some of them, such as Groth16 and Plonk [15, 10], are widely used and deployed.

These protocols also consider zero-knowledge property, zero-knowledge proofs have a long and fascinating history in computer science and cryptography. The concept was first introduced in a landmark paper by Goldwasser, Micali, and Rackoff in 1985 [13], which described a way for a prover to convince a verifier of the validity of a statement without revealing any additional information beyond the statement's truth. Later, Kilian and Micali [20, 22] developed the very first zero-knowledge proofs protocol.

In recent years, numerous follow-up works have built upon this foundational research, with a particular focus on improving the efficiency of the prover. A key area of recent advancement has been the development of protocols that achieve

linear prover time for arithmetic circuits over a large field. Several recent works, such as [31, 29, 34, 33, 3, 4, 14] have made significant strides in this area.

For Boolean circuits, recent work by [25, 18] has reduced the prover size to almost linear, with the provers running on the Boolean circuit model. These developments hold promise for the construction of zero-knowledge proof systems that are not only highly secure but also highly efficient. However, a key limitation of these protocols is that they rely on a linear time encodable linear code [27]. While this code can achieve a Boolean circuit size of $O(N)$, it still requires $O(N)$ RAM operations when considering the RAM computation model.

Thus we raise the following question:

Can construct a succinct proof for N-sized Boolean circuit on RAM machine using only $O(\frac{N}{\log N})$ RAM operations with $2^{-\lambda}$ soundness?

1.1 Our Results

In this work we have the following contributions:

1. We defined a new form of layered R1CS to denote the computation of layered circuits. With this new form, we can also describe the computation of Boolean circuits, of which each gate can be fixed in a single layer.

   Definition 1 (informal). *A layered R1CS is a arithmetic circuit where each gate is defined similar to a R1CS constraint. For each gate, it can take k inputs, let inputs be a vector x, the output of this gate will be*

   $$\langle \boldsymbol{a}\boldsymbol{x} \rangle * \langle \boldsymbol{b}\boldsymbol{x} \rangle + \langle \boldsymbol{c}\boldsymbol{x} \rangle$$

   where a, b, c are coefficient vectors of size k that are pre-determined when generating the circuit. Here k is a constant.

2. We purpose an interactive proof protocol for data-parallel Boolean circuits represented by layered R1CS. The proving time of this protocol is almost optimal to the computation of the Boolean circuit on RAM model.

   Theorem 1 (informal). *Assuming the layered R1CS circuit C is a data-parallel circuit with total size N, which is composed by $M = O(N^{0.1})$ copies of identical copies. There exists a succinct argument with soundness error $2^{-\lambda}$ that the prover runs in $O(\omega(1)\frac{N}{\log N} \log\log N)$ RAM operations. We also assumes the input size of the circuit is at most $O(\frac{N}{\log N})$ bits.*

   A key challenge in achieving efficient zero-knowledge proof systems is to remove the $\log N$ factor that arises when producing proofs. One approach to addressing this challenge is to make non-trivial use of the fact that a RAM word can store up to $O(\log N)$ bits. In our paper, we are based on the GKR protocol [12], and using techniques from [29].
   However, this is not a trivial task, as the GKR protocol used in many succinct proof systems will immediately turn any input (even Boolean input) into a

random field element, where each field element has more than λ bits. As a result, applying any GKR protocol directly to a Boolean circuit would result in $O(N)$ field operations.
To remove the $\log N$ factor, a modified GKR protocol has been developed based on the Libra protocol. This protocol uses a lookup table to compress $\log N$ field operations into one, with an approach similar to the Pippenger algorithm [24] or multi-scalar multiplication algorithms. These algorithms use a lookup table to skip $\log N$ bits in each operation.

1.2 Related works

Despite the line of work on linear time succinct proofs, there are many other works on zero-knowledge proofs. One line of work has focused on constructing non-succinct proofs, this line of work is derived from MPC protocols [19], and there are many recent developments [28, 9, 8, 32, 2], where these protocols achieves $O(N)$ on RAM machine which is highly practical.

Other line of work focusing on efficient verifier, these protocols are using a trusted-setup[16, 11, 23] or holographic-setup[6, 7] to speed up the verifier. Most trusted-setup protocols can construct a verifier with $O(1)$ verification time. For holographic-setup, the verifier will take at least $O(\log N)$ time.

2 Technical Overview

Our goal in this paper is to derive an interactive proof protocol for Boolean circuits with efficient proving time in terms of circuit computation.

Data-Parallel Circuit and Layered R1CS Model The framework for our solution is derived from the GKR protocol [12], which is an efficient instantiation of an interactive proof, particularly for layered circuits. It proves circuit computation by reducing the correctness from the current layer to the previous one. Formally, GKR indexes the layers from the output to the input, where the i-th layer has N_i gates. It describes the relationship between the $(i+1)$-th and i-th layers as follows:

$$\begin{aligned}\tilde{V}_i(\mathbf{z}) = \sum_{\mathbf{x},\mathbf{y}\in\{0,1\}^{n}_{i+1}} & \tilde{\mathsf{add}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})\left(\tilde{V}_{i+1}(\mathbf{x})+\tilde{V}_{i+1}(\mathbf{y})\right) \\ & + \tilde{\mathsf{mul}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})\left(\tilde{V}_{i+1}(\mathbf{x})\tilde{V}_{i+1}(\mathbf{y})\right)\end{aligned} \tag{1}$$

where $\mathbf{z} \in \{0,1\}^{n_i}$, and $\tilde{V}_i$ is the multilinear extension of the i-th layer. [29] shows that proving this summation can be reduced to two random evaluations on $\tilde{V}_{i+1}(\mathbf{x})$ and $\tilde{V}_{i+1}(\mathbf{y})$ through two stages of sumcheck protocols, which only costs $O(N_i+N_{i+1})$ field operations. Compared with Spartan [26] and Plonk [10], GKR only commits the values in the input layer but doesn't require computing the commitments of intermediate layers. Therefore, it has better performance for circuits with small public input and witnesses.

However, when dealing with Boolean circuits, linear complexity alone is not efficient enough, as the length of the field elements needs to be taken into consideration. To reduce the proving time, one intuition is to exploit the fact that all circuit wires are Boolean values, and allow the prover to aggregate as many identical terms as possible. In this paper, we therefore focus on the data-parallel circuit, assuming that the layer can be divided into $M = 2^m$ identical sub-circuits. The relationship equation then becomes:

$$\begin{aligned}\tilde{V}_i(\mathbf{z}\|\mathbf{t}) = \sum_{\mathbf{s}\in\{0,1\}^m} \tilde{\beta}_m(\mathbf{s},\mathbf{t}) \Bigg(& \sum_{\mathbf{x},\mathbf{y}\in\{0,1\}^{n_{i+1}-m}} \tilde{\mathsf{add}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y}) \left(\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) + \tilde{V}_{i+1}(\mathbf{y}\|\mathbf{s})\right) \\ & + \tilde{\mathsf{mul}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y}) \left(\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s})\tilde{V}_{i+1}(\mathbf{y}\|\mathbf{s})\right)\Bigg) \\ = & \sum_{\mathbf{s}\|\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{\beta}_m(\mathbf{t},\mathbf{s}) \left(\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) f(\mathbf{x}\|\mathbf{s}) + g(\mathbf{x}\|\mathbf{s})\right)\end{aligned}$$

where $f(\mathbf{x}\|\mathbf{s}) = \sum_{\mathbf{y}\in\{0,1\}^{n_{i+1}-m}} \tilde{\mathsf{add}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y}) + \tilde{\mathsf{mul}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})\tilde{V}_{i+1}(\mathbf{y}\|\mathbf{s})$, as well as $g(\mathbf{x}\|\mathbf{s}) = \sum_{\mathbf{y}\in\{0,1\}^{n_{i+1}-m}} \tilde{\mathsf{add}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})\tilde{V}_{i+1}(\mathbf{y}\|\mathbf{s})$. As mentioned earlier, we aim to exploit the special structure of the Boolean circuit to reduce computation. However, neither $f(\mathbf{x}|\mathbf{s})$ nor $g(\mathbf{x}|\mathbf{s})$ have any special structures. Therefore, we propose a new way to represent the relationship between adjacent layers, which is called Layered R1CS (LR1CS).

For each pair of adjacent layers, the equation is:

$$\begin{aligned}&\tilde{V}_i(\mathbf{z}) \\ &= \sum_{\mathbf{y}\in\{0,1\}^{n_i}} \tilde{\beta}_{n_i}(\mathbf{z},\mathbf{y}) \cdot \Bigg(\left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}) \right) \\ &+ \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}) \right) \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}) \right) \Bigg)\end{aligned} \tag{2}$$

Moreover, for the data-parallel circuit, we have the following equation:

$$\begin{aligned}&\tilde{V}_i(\mathbf{z}\|\mathbf{t}) \\ &= \sum_{\mathbf{y}\|\mathbf{s}\in\{0,1\}^{n_i}} \tilde{\beta}_m(\mathbf{t},\mathbf{s}) \cdot \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) \Bigg(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \\ &+ \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \right) \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \right) \Bigg) \\ &= \sum_{\mathbf{y}\|\mathbf{s}\in\{0,1\}^{n_i}} \tilde{\beta}_{n_i}(\mathbf{z}\|\mathbf{t},\mathbf{y}\|\mathbf{s}) \left(h(\mathbf{y}\|\mathbf{s}) + f(\mathbf{y}\|\mathbf{s})g(\mathbf{y}\|\mathbf{s})\right)\end{aligned}$$

Here $h(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s})$ and we define f, g similarly. Since we are dealing with a Boolean circuit, the evaluation of h, f, g on

Boolean hypercube will always be a Boolean value. Additionally, if it is a classical Boolean circuit with NAND gates only, then $\tilde{A}_{i+1}, \tilde{B}_{i+1}, \tilde{C}_{i+1}$ will only have 1 non-zero entry for each row $\mathbf{y}$, because the NAND gate only takes 2 inputs for multiplication and 1 constant input for addition. So for simplicity, we will treat evaluations of f, g, h as Boolean.

To prove this equation, we can divide the process into two stages of sumcheck protocols, the first stage will do sumcheck on variable s, focusing on the following summation:

$$\sum_{\mathbf{s}\in\{0,1\}^m} \tilde{\beta}_m(\mathbf{t},\mathbf{s}) \left(\sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) \left(h(\mathbf{y}\|\mathbf{s}) + f(\mathbf{y}\|\mathbf{s})g(\mathbf{y}\|\mathbf{s})\right) \right)$$

We observe that $f(\mathbf{y}|\mathbf{s})$ and $g(\mathbf{y}|\mathbf{s})$ have binary values on the hypercube domain. Intuitively, we can use a Pippenger-style summation to compute the sum of Boolean values by precomputing a small table and looking up a bit-vector of size $O(\log N)$ in the table for each iteration. Since each iteration processes $O(\log N)$ bits, there will be $O(\frac{N}{\log N})$ iterations. By setting the window size to $w = 0.1 \log N$, we can obtain an algorithm that performs $\frac{2^{n_i}}{w}$ field additions for each sum-check query. After $\log w$ rounds of sumcheck, the domain size is $O(\frac{2^{n_i}}{w})$, which is sufficiently small for us to proceed using the linear-time GKR algorithm to deal with the remaining rounds.

Now we take a detour to review the process of the Pippenger algorithm and show a variant of Pippenger to deal with addition gates.

2.1 Pippenger's algorithm

Consider the following problem: there is an field element vector $\boldsymbol{e}$ of length l, where addition on two field elements takes T_{add} time, and B is a $m \times l$ Boolean vector. Compute the matrix multiplication $B\boldsymbol{e}$. For simplicity, we assume that the length of each vector l is a power of 2.

Naive computation of this sum will incur $O(m \times l)$ additions of field elements in $\boldsymbol{e}$. In Pippenger's algorithm, a table is preprocessed, defined as follows:

1. Let $w = 0.1 \log l$, be the length of window.
2. For each bit-vector $\boldsymbol{i} \in \{0,1\}^w$ and $j \in [2^{0.9 \log l}]$, compute the table $\mathsf{T}[j][\boldsymbol{i}] := \langle \boldsymbol{i}, \boldsymbol{e}[j : j + w] \rangle$.

The initialization of this table takes $O(wl)$. We can compute the matrix multiplication as follows:

1. For each $i \in [m]$, compute the i-th entry of the result using inner product $\langle B[i][:], \boldsymbol{e} \rangle$.
2. To compute the inner product, we can use the table to speed up:
   (a) For $i \in [\frac{l}{w}]$: add $\mathsf{T}[i][\boldsymbol{e}[i*w : i*w+w]]$ into the inner-product summation.

(b) Lookup table takes $O(1)$ time on RAM machine, so the overall complexity is $O(\frac{l}{w}T_{add})$.

3. Since there are m inner-products, so it will take $O(\frac{ml}{w}T_{add})$ time.

In our protocol, the field element will be degree 1 polynomials where coefficients are field element, so $T_{add} = O(\omega(1))$.

2.2 Using Pippenger to deal with addition gates

If we ignore the multiplication part, the sumcheck equation becomes:

$$\sum_{\mathbf{s}\in\{0,1\}^m} \tilde{\beta}_m(\mathbf{t},\mathbf{s}) \left(\sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) h(\mathbf{y}\|\mathbf{s}) \right)$$

For i-th round of the sumcheck, we are required to compute the following polynomial:

$$\sum_{\mathbf{s}\in\{0,1\}^{m-1}} \tilde{\beta}_{m-i}(\mathbf{t}[0:m-i],\mathbf{s}[0:m-i])\tilde{\beta}(t[m-i:],x|\mathbf{r}_s)\times$$

$$\left(\sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y})\tilde{\beta}_{i-1}(\mathbf{r}_s,\mathbf{s}[m-i:]) h(\mathbf{y}\|\mathbf{s}[0:m-i]|x|\mathbf{s}[m-i:]) \right)$$

Here $\mathbf{r}_s$ is the randomness sent from verifier by the i-th round, so it has length $i-1$.

For each concrete value of s, the outer part $\tilde{\beta}_{m-i}(\mathbf{t}[0:m-i],\mathbf{s}[0:m-i])\tilde{\beta}(t[m-i:],x|\mathbf{r}_s)$ is easy to compute so we skip this part. However the inner part:

$$\left(\sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y})\tilde{\beta}_{i-1}(\mathbf{r}_s,\mathbf{s}[m-i:]) h(\mathbf{y}\|\mathbf{s}[0:m-i]|x|\mathbf{s}[m-i:]) \right)$$

is not so straightforward. We first remove the summation, and try to extract the Boolean vector and the field element vector:

$$\tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y})\tilde{\beta}_{i-1}(\mathbf{r}_s,\mathbf{s}[m-i:]) h(\mathbf{y}\|\mathbf{s}[0:m-i]|x|\mathbf{s}[m-i:])$$

Here $\tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}), \tilde{\beta}_{i-1}(\mathbf{r}_s,\mathbf{s}[m-i:])$ are field elements that is easy to compute. $h(\mathbf{y}\|\mathbf{s}[0:m-i]|x|\mathbf{s}[m-i:])$ is a degree 1 polynomial. We use $h(\mathbf{y}|x)$ as a short-hand to $h(\mathbf{y}\|\mathbf{s}[0:m-i]|x|\mathbf{s}[m-i:])$.

For $x=0$, the evaluation is equal to $h(\mathbf{y}\|0)$, and for $x=1$, the evaluation is equal to $h(\mathbf{y}\|1)$. So the polynomial $h(\mathbf{y}|x) = (h(\mathbf{y}|1) - h(\mathbf{y}|0))x + h(\mathbf{y}|0)$. And we incorporate $\tilde{\beta}$ functions into our polynomial, the polynomial becomes:

$$h(\mathbf{y}|x) = \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y})\tilde{\beta}_{i-1}(\mathbf{r}_s, \mathbf{s}[m-i:])((h(\mathbf{y}|1) - h(\mathbf{y}|0))x + h(\mathbf{y}|0))$$

To compute the summation over this polynomial, we can compute the coefficient individually as a inner-product between field elements and Boolean matrix using Pippenger's algorithm. The Boolean matrix has shape $2^{m-1} \times 2^{n_i}$ and the field vector has length $O(2^{n_i})$, so the complexity will be $O(\frac{2^{m+n_i}}{w})$ field additions.

2.3 Difficulties in using Pippenger's algorithm and MSM

Unfortunately, Pippenger's algorithm will stop working when we deal with a multiplication gate. Consider the following multiplication gate:

$$\sum_{\mathbf{s}\in\{0,1\}^m} \tilde{\beta}_m(\mathbf{t},\mathbf{s}) \left(\sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) f(\mathbf{y}\|\mathbf{s}) g(\mathbf{y}\|\mathbf{s}) \right)$$

In the i-th round of the sumcheck protocol, we need to compute the polynomial of the following form:

$$\sum_{\mathbf{s}[i:]\in\{0,1\}^{m-1}} \tilde{\beta}_m(\mathbf{t}, \mathbf{s}[:m-i]||x||\mathbf{r}_s) \times \sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \Big(\tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) \times$$

$$\tilde{\beta}_{i-1}(s[m-i:], \mathbf{r}_s) f(\mathbf{y}\|\mathbf{s}[:m-i]||x||\mathbf{s}[m-i:]) \times$$

$$\tilde{\beta}_{i-1}(s[m-i:], \mathbf{r}_s) g(\mathbf{y}\|\mathbf{s}[:m-i]||x||\mathbf{s}[m-i:]) \Big)$$

Here, the inner-summation becomes a univariate quadratic polynomial with two different Boolean vectors f, g. We solve this problem in Algorithm 4 by computing a 2-D version of Pippenger's algorithm where we couple the lookup table for f and g. We extend the table from $\mathsf{T}[i,j]$ to $\mathsf{T}[i_0, i_1, j]$. Adding one more dimension to compute the inner-product between f and g.

2.4 The remaining part

Multi-scalar multiplication (MSM) for computing the inner product between fields and bits. In order to transition from our Pippenger-style algorithm to the linear-time GKR algorithm, we need to initialize the bookkeeping table used in [29]. The two parties run a sumcheck protocol for variable $\mathbf{y}$ on the following equation:

$$\tilde{\beta}_m(\mathbf{t}\mathbf{r}_s) \sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}_{n_i-m}(\mathbf{z},\mathbf{y}) \left(h(\mathbf{y}\|\mathbf{r}_s) + f(\mathbf{y}\|\mathbf{r}_s) g(\mathbf{y}\|\mathbf{r}_s)\right)$$

Although the problem size of the sumcheck protocol is only $\frac{N_{i+1}}{M}$, the initialization, e.g., computing the book-keeping table as in [29], still requires $O(N_{i+1})$ field additions. To deal with this problem, we observe that the most costly part

is computing the inner product between several field vectors and bit vectors. Since the inner product is in the same form as MSM, we can use the Pippenger algorithm [24] to accelerate the computation. We present the algorithm in Algorithm 1.

3 Preliminaries

3.1 Notation

In this paper, we use the following notations. For a d-variate polynomial $f(x_0, \ldots, x_{d-1}) : \mathbb{F}^d \to \mathbb{F}$, we use $\mathbf{x} = (x_0, \ldots, x_{d-1})$ to denote the vector of variables in the domain. After assigning those variables as random elements, we denote them as $\mathbf{r}_x$. Given a function $f : \{0,1\}^d \to \{0,1\}$ with the domain d-dimentional Boolean hypercube, we use $\tilde{f} : \mathbb{F}^d \to \mathbb{F}$ as the unique multilinear extension (MLE) (see Definition 4), satisfying $\tilde{f}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^d$. There is a commonly-used function $\tilde{\beta}_n(\mathbf{x}, \mathbf{y}) = \prod_{i=0}^{n-1} x_i y_i + (1 - x_i)(1 - y_i)$, which is the MLE of the following function

$$\beta_n(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \mathbf{x} = \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We use the LR1CS to describe a circuit. In this form, the indices of layers are numbered from the output to the input. $\tilde{V}_i(\cdot)$ is the MLE corresponding to the values in the i-th layer. $\tilde{A}_i(\cdot)$, $\tilde{B}_i(\cdot)$ and $\tilde{C}_i(\cdot)$ together form the computation of the i-th layer. When it's for data-parallel circuits, we use M to denote the number of copies of the sub-circuits. In the notation of $\tilde{V}_i(\mathbf{x}\|\mathbf{s})$, the variable $\mathbf{s}$ is to index the sub-circuit copy, and the variable $\mathbf{x}$ is to specify a specific wire in the sub-circuit.

3.2 Finite field

Throughout the paper, we assume that all fields are prime fields $\mathbb{F}_p$, where p is a prime number satisfying $p \leq 2^\lambda$ and $\lambda = O(\omega(1) \log N)$. In practice, we use 128-bit or 256-bit primes. Such prime fields must satisfy the following performance requirements:

1. Storage: A field element can be stored using $O(\omega(1))$ RAM words.
2. Addition: Addition can be done within $O(\omega(1))$ RAM operations.
3. Multiplication: Multiplication can be done within $O(\omega^2(1))$ RAM operations.

These requirements ensure that field operations can be performed efficiently, which is crucial for the overall performance of our protocols.

3.3 Interactive Proofs and Zero-knowledge Arguments

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 in previous rounds. We phrase this in terms of $\mathcal{P}$ trying to convince $\mathcal{V}$ that $f(x) = 1$. The proof system is interesting only when the running time of $\mathcal{V}$ is less than the time of directly computing the function f. We formalize interactive proofs in the following:

Definition 2. *Let f be a Boolean 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 $f(x) = 1$ it holds that* $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = \mathit{accept}] = 1$.
- **ϵ-Soundness.** *For any x with $f(x) \neq 1$ and any $\mathcal{P}^*$ it holds that* $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle = \mathit{accept}] \leq \epsilon$

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 $(x; w) \in \mathcal{R}$ for some input x. We focus on arguments of knowledge that 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 key pk and the verification key vk. Formally, consider the definition below, where we assume $\mathcal{R}$ 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 for $\mathcal{R}$ if the following holds.*

- ***Correctness***. *For every $(\mathsf{pk}, \mathsf{vk})$ output by $\mathcal{G}(1^\lambda)$ and $(x, w) \in \mathcal{R}$,*

$$\langle \mathcal{P}(\mathsf{pk}, w), \mathcal{V}(\mathsf{vk}) \rangle (x) = \mathsf{accept}$$

- ***Soundness***. *For any PPT prover $\mathcal{P}$, there exists a PPT extractor ε such that for every $(\mathsf{pk}, \mathsf{vk})$ output by $\mathcal{G}(1^\lambda)$ and any x, it holds that*

$$\Pr[\langle \mathcal{P}(\mathsf{pk}), \mathcal{V}(\mathsf{vk}) \rangle(x) = \mathsf{accept} \wedge (x, w) \notin \mathcal{R} | w \leftarrow \varepsilon(\mathsf{pk}, x)] \leq \mathsf{negl}(\lambda)$$

- ***Zero knowledge***. *There exists a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, $(x; w) \in \mathcal{R}$, it holds that*
  $\Pr\left[\langle \mathcal{P}(\mathsf{pk}, w), \mathcal{A} \rangle(x) = \mathsf{accept} : (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{G}(1^\lambda); (x, w) \leftarrow \mathcal{A}(z, \mathsf{pk}, \mathsf{vk})\right] =$
  $\Pr\left[\langle \mathcal{S}(\mathsf{trap}, z, \mathsf{pk}), \mathcal{A} \rangle(x) = \mathsf{accept} : (\mathsf{pk}, \mathsf{vk}, \mathsf{trap}) \leftarrow \mathcal{S}(1^\lambda); (x, w) \leftarrow \mathcal{A}(z, \mathsf{pk}, \mathsf{vk})\right]$

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

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

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

$$f_1(x_{\ell-1}) \stackrel{def}{=} \sum_{b_0,\ldots,b_{\ell-2}\in\{0,1\}} f(b_0,\ldots,b_{\ell-2},x_{\ell-1}),$$

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

$$f_i(x_{\ell-i}) \stackrel{def}{=} \sum_{b_0,\ldots,b_{\ell-i-1}\in\{0,1\}} f(b_0,\ldots,b_{\ell-i-1},x_{\ell-i},r_{\ell-i+1},\ldots,r_{\ell-1}),$$

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

$$f_\ell(x_0) \stackrel{def}{=} f(x_0,r_1,\ldots,r_{\ell-1}),$$

  $\mathcal{V}$ checks $f_{\ell-1}(r_0) = f_\ell(0) + f_\ell(1)$. The verifier generates a random challenge $r_0 \in \mathbb{F}$. Given oracle access to an evaluation $f(r_0,r_1,\ldots,r_{\ell-1})$ of f, $\mathcal{V}$ will accept if and only if $f_\ell(r_0) = f(r_0,r_1,\ldots,r_{\ell-1})$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

3.4 GKR Protocol

In [12], Goldwasser et al. proposed an efficient interactive proof protocol for layered arithmetic circuits, from which we follow the framework to design our new protocol and is referred to as the *GKR* protocol. We present the detailed protocol here.

Sumcheck Protocol. The sumcheck problem is a fundamental problem that has various applications. The problem is to sum a polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$ on the binary hypercube

$$\sum\nolimits_{b_0,b_1,\ldots,b_{\ell-1}\in\{0,1\}} f(b_0,b_1,...,b_{\ell-1}).$$

Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_0,\ldots,b_{\ell-1}$. Lund et al. [21] 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 $O(d\ell)$, where d is the variable degree of f, as in each round, $\mathcal{P}$ sends a univariate polynomial of one variable in f, which $d+1$ points can uniquely define. The verifying time of the protocol is $O(d\ell)$. The proving time depends on the degree and the sparsity of f. For example, if f is a multilinear polynomial, the proving time is $O\left(2^\ell\right)$. The sumcheck protocol is complete and sound with $\epsilon = \frac{d\ell}{|\mathbb{F}|}$.

Definition 4 (Multilinear 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 $\tilde{V}(\mathbf{b}) = V(\mathbf{b})$ for all $\mathbf{b} \in \{0,1\}^\ell$.*

$\tilde{V}$ can be expressed as:

$$\tilde{V}(\mathbf{x}) = \sum\nolimits_{\mathbf{b}\in\{0,1\}^\ell} \prod\nolimits_{i=0}^{\ell-1} [((1-x_i)(1-b_i) + x_i b_i) \cdot V(\mathbf{b})]$$

where $\mathbf{x} = (x_0, \ldots, x_{\ell-1})$ and $\mathbf{b} = (b_0, \ldots, b_{\ell-1})$.

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

Using the sumcheck protocol as a building block, Goldwasser et al. [12] showed an interactive proof protocol for layered arithmetic circuits, which is commonly called GKR protocol.

Notation. Before describing the GKR protocol, we introduce some additional notations. We denote the number of gates in the i-th layer as S_i and let $s_i = \log S_i$. (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 the b-th gate in the i-th layer, 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, for $1 \le i \le d$, we define two additional functions $\mathsf{add}_i, \mathsf{mul}_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, referred as *wiring predicates* in the literature. add_i (mul_i) takes one gate label $\mathbf{z} \in \{0,1\}^{s_{i-1}}$ in the $(i-1)$-th layer with two gate labels $\mathbf{x}, \mathbf{y} \in \{0,1\}^{s_i}$ in the i-th layer, and outputs 1 if and only if Gate $\mathbf{z}$ is an addition (multiplication) gate that takes the output of Gate $\mathbf{x}$ and Gate $\mathbf{y}$ as input. With these definitions, V_i can be written as follows:

$$\begin{aligned} V_i(\mathbf{z}) = \sum_{\mathbf{x},\mathbf{y}\in\{0,1\}^{s_{i+1}}} & (\mathsf{add}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})(V_{i+1}(\mathbf{x}) + V_{i+1}(\mathbf{y})) \\ & + \mathsf{mul}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})(V_{i+1}(\mathbf{x})V_{i+1}(\mathbf{y}))) \end{aligned} \tag{4}$$

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

$$\begin{aligned} \tilde{V}_i(\mathbf{z}) =& \sum\nolimits_{\mathbf{x},\mathbf{y}\in\{0,1\}^{s_{i+1}}} f_{i+1}(\mathbf{x},\mathbf{y}) \\ =& \sum\nolimits_{\mathbf{x},\mathbf{y}\in\{0,1\}^{s_{i+1}}} (\tilde{\mathsf{add}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})(\tilde{V}_{i+1}(\mathbf{x}) + \tilde{V}_{i+1}(\mathbf{y})) \\ & + \tilde{\mathsf{mul}}_{i+1}(\mathbf{z},\mathbf{x},\mathbf{y})(\tilde{V}_{i+1}(\mathbf{x})\tilde{V}_{i+1}(\mathbf{y}))), \end{aligned} \tag{5}$$

where $\mathbf{z}$ is a vector of variables and can be assigned from the domain $\mathbb{F}^{s_i}$.

Protocol. With Equation 5, 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 $\tilde{V}_0(\mathbf{r}_{z,0})$ for a random $\mathbf{r}_{z,0} \in \mathbb{F}^{s_0}$. $\mathcal{V}$ and $\mathcal{P}$ then invoke a sumcheck protocol on Equation 5 with all $0 \leq i < d$. As described in Section 3.4, at the end of the i-th sumcheck protocol, $\mathcal{V}$ needs $f_{i+1}(\mathbf{r}_{x,i+1}, \mathbf{r}_{y,i+1})$ with random $\mathbf{r}_{x,i+1}, \mathbf{r}_{y,i+1} \in \mathbb{F}^{s_{i+1}}$ to complete the proof. Hence, $\mathcal{V}$ computes $\tilde{\mathsf{add}}_{i+1}(\mathbf{r}_{x,i+1}, \mathbf{r}_{y,i+1})$ and $\tilde{\mathsf{mul}}_{i+1}(\mathbf{r}_{x,i+1}, \mathbf{r}_{y,i+1})$ locally (they only depend on the wiring pattern of the circuit, but not on the values), asks $\mathcal{P}$ for $\tilde{V}_{i+1}(\mathbf{r}_{x,i+1})$ and $\tilde{V}_{i+1}(\mathbf{r}_{y,i+1})$ and computes $f_{i+1}(\mathbf{r}_{x,i+1}, \mathbf{r}_{y,i+1})$. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduce a claim about the i-th layer to two claims about values in the $(i+1)$-th layer.

Combining two claims: random linear combination. In [5], Chiesa et al. proposed an approach using random linear combinations to deal with the two claims. Upon receiving the two claims $\tilde{V}_i(\mathbf{r}_{x,i})$ and $\tilde{V}_i(\mathbf{r}_{y,i})$, $\mathcal{V}$ selects $\alpha_i, \beta_i \in \mathbb{F}$ randomly and computes $\alpha_i \tilde{V}_i(\mathbf{r}_{x,i}) + \beta_i \tilde{V}_i(\mathbf{r}_{y,i})$. Based on Equation 5, this random linear combination can be written as

$$
\begin{aligned}
&\alpha_i \tilde{V}_i(\mathbf{r}_{x,i}) + \beta_i \tilde{V}_i(\mathbf{r}_{y,i}) \\
&= \alpha_i \sum_{\substack{\mathbf{u}\in\{0,1\}^{s_{i+1}} \\ \mathbf{v}\in\{0,1\}^{s_{i+1}}}} \Big(\tilde{\mathsf{add}}_{i+1}(\mathbf{r}_{x,i}, \mathbf{u}, \mathbf{v})(\tilde{V}_{i+1}(\mathbf{u}) + \tilde{V}_{i+1}(\mathbf{v})) \\
&\qquad\qquad + \tilde{\mathsf{mul}}_{i+1}(\mathbf{r}_{x,i}, \mathbf{u}, \mathbf{v})(\tilde{V}_{i+1}(\mathbf{u})\tilde{V}_{i+1}(\mathbf{v})) \Big) \\
&+ \beta_i \sum_{\substack{\mathbf{u}\in\{0,1\}^{s_{i+1}} \\ \mathbf{v}\in\{0,1\}^{s_{i+1}}}} \Big(\tilde{\mathsf{add}}_{i+1}(\mathbf{r}_{y,i}, \mathbf{u}, \mathbf{v})(\tilde{V}_{i+1}(\mathbf{u}) + \tilde{V}_{i+1}(\mathbf{v})) \\
&\qquad\qquad + \tilde{\mathsf{mul}}_{i+1}(\mathbf{r}_{y,i}, \mathbf{u}, \mathbf{v})(\tilde{V}_{i+1}(\mathbf{u})\tilde{V}_{i+1}(\mathbf{v})) \Big) \\
&= \sum_{\substack{\mathbf{u}\in\{0,1\}^{s_{i+1}} \\ \mathbf{v}\in\{0,1\}^{s_{i+1}}}} ((\alpha_i \tilde{\mathsf{add}}_{i+1}(\mathbf{r}_{x,i}, \mathbf{u}, \mathbf{v}) + \beta_i \tilde{\mathsf{add}}_{i+1}(\mathbf{r}_{y,i}, \mathbf{u}, \mathbf{v}))(\tilde{V}_{i+1}(\mathbf{u}) + \tilde{V}_{i+1}(\mathbf{v})) \\
&\qquad\qquad + (\alpha_i \tilde{\mathsf{mul}}_{i+1}(\mathbf{r}_{x,i}, \mathbf{u}, \mathbf{v}) + \beta_i \tilde{\mathsf{mul}}_{i+1}(\mathbf{r}_{y,i}, \mathbf{u}, \mathbf{v}))(\tilde{V}_{i+1}(\mathbf{u})\tilde{V}_{i+1}(\mathbf{v}))) \qquad (6)
\end{aligned}
$$

$\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 6 instead of Equation 5. Then at the end of each sumcheck protocol, $\mathcal{V}$ still receives two claims about $\tilde{V}_{i+1}$ and proceeds to a previous layer recursively until the input layer.

From the result of [29], we have the following theorem:

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

3.5 Rank-1 Constraint Satisfiability

In this section, we define the Rank-1 Constraint Satisfiability (R1CS) instance and problem.

Definition 5 (R1CS instance). *An R1CS instance is a tuple* $(\mathbb{F}, A, B, C, io, N, N_g)$*, where* io *denotes the public input and output of the instance,* $A, B, C \in \mathbb{F}^{N \times N}$*, where* $N \geq |io| + 1$ *and there are at most* N_g *non-zero entries in each matrix.*

In the definition above, we assume matrices A, B, and C are the square matrix for simplicity and $N = O(N_g)$.

Definition 6 (R1CS). *An R1CS instance* $(\mathbb{F}, A, B, C, io, N, N_g)$ *is said to be satisfiable if there exists a witness* $\mathsf{w} \in \mathbb{F}^{N-|io|-1}$ *such that* $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$*, where* $z = (io, \mathsf{w}, 1)$*,* $\cdot$ *is the matrix-vector product, and* $\circ$ *is the Hadamard (entry-wise) product.*

Definition 7. *For an R1CS instance* $\mathbb{x} = (\mathbb{F}, A, B, C, io, N, N_g)$ *and a purposed witness* $\mathsf{w} \in \mathbb{F}^{N-|io|-1}$*, we define:*

$$\mathtt{Sat}_{\mathtt{R1CS}}(\mathbb{x}, \mathsf{w}) = \begin{cases} 1 & (A \cdot (io, \mathsf{w}, 1)) \circ (B \cdot (io, w, 1)) = (C \cdot (io, \mathsf{w}, 1)) \\ 0 & otherwise \end{cases}$$

The set of satisfiable R1CS instances can be denoted as:

$$\mathcal{R}_{\mathtt{R1CS}} = \{\langle (\mathbb{F}, A, B, C, io, N, N_g), w \rangle : \mathtt{Sat}_{\mathtt{R1CS}}((\mathbb{F}, A, B, C, io, N, N_g), \mathsf{w}) = 1\}$$

Definition 8. *For a given R1CS instance* $\mathbb{x} = (\mathbb{F}, A, B, C, io, N, N_g)$*, the NP statement that* $\mathbb{x}$ *is satisfiable (i.e.,* $\langle \mathbb{x}, \cdot \rangle \in \mathcal{R}_{\mathtt{R1CS}}$*) is of size* $O(N_g)$

4 Interactive Proof Protocol for Layered R1CS (LR1CS)

In this section, we present a formal definition of LR1CS and state the main theorem for realizing a new interactive proof protocol through this form. Specifically, in Section 4.1, we provide definitions of LR1CS instance and LR1CS language to help formalize the problem. In Section 4.2, we show that by using the form of LR1CS, we can construct an interactive proof protocol with efficient proving time.

4.1 Definition of LR1CS

Here we give the definitions of LR1CS instance and LR1CS language as follows:

Definition 9 (LR1CS instance). *An LR1CS instance is a tuple*

$$(\mathbb{F}, d, N_0, V_d, V_0, \{A_i, B_i, C_i, N_i, K_i\}_{1 \le i \le d})$$

where V_d and V_0 denote the public input and output of the instance with size N_d and N_0 separately, N_i is the number of wires in the i-th layer, $A_i, B_i, C_i \in \mathbb{F}^{(N_{i-1}+1)\times(N_i+1)}$ describing the computation for the i-th layer and there are at most K_i non-zero entries in each matrix.

In this paper, we always assume $K_i = O(N_{i-1})$. With the definition above, we define LR1CS language as follows:

Definition 10 (LR1CS). *For an LR1CS instance*

$$\mathbb{x} = (\mathbb{F}, d, N_0, V_d, V_0, \{A_i, B_i, C_i, N_i, K_i\}_{1 \le i \le d})$$

if there exists witness $w = (V_1(\mathbf{x}), \ldots, V_{d-1}(\mathbf{x}))$, for all i, $V_i(\mathbf{x}) : \{0,1\}^{\log N_i} \to \mathbb{F}$, and for $0 \le i < d, \mathbf{z} \in \{0,1\}^{\log N_i}$ it satisfies the following equations

$$\begin{aligned}
&\tilde{V}_i(\mathbf{z}) \\
&= \sum_{\mathbf{y} \in \{0,1\}^{n_i}} \tilde{\beta}_{n_i}(\mathbf{z}, \mathbf{y}) \cdot \left(\left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{C}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \right. \\
&\left. + \left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{A}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{B}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \right)
\end{aligned} \tag{7}$$

$\mathsf{Sat}_{\mathsf{LR1CS}}(\mathbb{x}, w) = 1$, *otherwise* 0. *The set of satisfiable LR1CS instances can be denoted as:*

$$\mathcal{R}_{\mathsf{LR1CS}} = \left\{ \mathbb{x} \in \left\{ (\mathbb{F}, d, N_0, V_d, V_0, \{A_i, B_i, C_i, N_i, K_i\}_{1 \le i \le d}) \right\} : \mathsf{Sat}_{\mathsf{LR1CS}}(\mathbb{x}, \mathsf{w}) = 1 \right\}$$

Use LR1CS to denote Boolean circuits. We can easily see the completeness of LR1CS to cover arithmetic circuits consist of addition and multiplication gates. As for Boolean circuits, Lemma 1.1 in [17] has shown that any n-ary boolean operation can be represented as a multilinear polynomial with n variables. Since one-layer LR1CS can describe any multilinear polynomial with two variables, it can also represent a layer of Boolean gates. We provide several examples to write Boolean gates in LR1CS form in Table 4.1. In our protocol, to keep small proving complexity, we assume that the Boolean circuit only consists of AND gates and NOT gates.

Gate	Arithmetic form	LR1CS (in: x, y, out: z)
$\mathsf{XOR}(a, b)$	$a + b - 2ab$	$C(z, x) = C(z, y) = 1$ $A(z, x) = -2$ $B(z, y) = 1$
$\mathsf{OR}(a, b)$	$a + b - ab$	$C(z, x) = C(z, y) = 1$ $A(z, x) = -1$ $B(z, y) = 1$
$\mathsf{AND}(a, b)$	$a \cdot b$	$A(z, x) = 1$ $B(z, y) = 1$
$\mathsf{NOT}(a, \_)$	$1 - a$	$C(z, N) = 1$ (constant entry) $C(z, x) = -1$
$\mathsf{NAND}(a, b)$	$1 - ab$	$C(z, N) = 1$ $A(z, x) = -1$ $B(z, x) = 1$

Table 1. Examples of writing boolean gates in LR1CS form

4.2 Interactive proof Protocol

Now we are aiming to design an interactive proof protocol for LR1CS. based on the Definition 9 and Definition 10. Given an LR1CS instance $\mathbb{x}$, we have the following theorem:

Theorem 3. *Let $\mathbb{x} \in \mathcal{R}_{\mathsf{R1CS}}$, assume $N_\Sigma = \sum_{i=0}^{d} N_i$, there is an interactive proof protocol with soundness $O(d \log N_\Sigma / |\mathbb{F}|)$. It uses $O(d \log N_\Sigma)$ rounds of interaction and the running time of the prover $\mathcal{P}$ is $O(N_\Sigma)$. Let T be the time to evaluate all $\tilde{A}_i$, $\tilde{B}_i$ and $\tilde{C}_i$ at the corresponding random points, the running time of $\mathcal{V}$ is $O(N_0 + N_d + d log N_\Sigma + T)$.*

Proof. Similar to the GKR protocol, we prove the correctness of LR1CS instance $\mathbb{x}$ layer by layer. At the beginning of the protocol, the verifier $\mathcal{V}$ sends a random element $\mathbf{r}_{z,0}$ to the prover $\mathcal{P}$. Next, both $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{V}_0(\mathbf{r}_{z,0})$. After that, for $0 \leq i < d$, $\mathcal{P}$ and $\mathcal{V}$ prove the computation for the i-th layer through the following multilinear equation:

$$\begin{aligned}
&\tilde{V}_i(\mathbf{r}_{z,i}) \\
&= \sum_{\mathbf{y} \in \{0,1\}^{n_i}} \tilde{\beta}_{n_i}(\mathbf{r}_{z,i}, \mathbf{y}) \cdot \left(\left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{C}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \right. \\
&\left. + \left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{A}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \left(\sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} \tilde{B}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x}) \right) \right)
\end{aligned} \tag{8}$$

where $n_{i+1} = \log N_{i+1}$. With a similar idea from the proof of Theorem 2, $\mathcal{P}$ and $\mathcal{V}$ go through the following stages:

1. $\mathcal{P}$ **and** $\mathcal{V}$ **prove a rowcheck relation**. In this stage, the prover and the verifier run a sumcheck protocol on the following equation:

$$\tilde{V}_i(\mathbf{r}_{z,i}) = \sum_{\mathbf{y}\in\{0,1\}^{n_i}} f(\mathbf{y})g(\mathbf{y}) + h(\mathbf{y}) \tag{9}$$

where $f(\mathbf{y})$, $g(\mathbf{y})$ and $h(\mathbf{y})$ are computed from the following equations:

$$f(\mathbf{y}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_i(\mathbf{x}) \tag{10}$$

$$g(\mathbf{y}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_i(\mathbf{x}) \tag{11}$$

$$h(\mathbf{y}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_i(\mathbf{x}) \tag{12}$$

Computing each bookkeeping table requires $O(N_i)$ in the proving time since each matrix has $O(N_i)$ non-zero entries. After the sumcheck protocol of $O(N_{i+1}$ proving time, $\mathcal{V}$ receives $f(\mathbf{r}_y)$, $g(\mathbf{r}_y)$ and $h(\mathbf{r}_y)$ from $\mathcal{P}$.

2. $\mathcal{P}$ **and** $\mathcal{V}$ **prove a lincheck relation**. In this stage, $\mathcal{V}$ first sends $(\gamma_{i+1}, \delta_{i+1}, \eta_{i+1})$ to $\mathcal{P}$. Then the two parties launch a sumcheck protocol for the following equation:

$$\begin{aligned}
&\gamma_{i+1}f(\mathbf{r}_y) + \delta_{i+1}g(\mathbf{r}_y) + \eta_{i+1}h(\mathbf{r}_y) \\
&= \gamma_{i+1} \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{A}_{i+1}(\mathbf{r}_y,\mathbf{x})\tilde{V}_i(\mathbf{x}) \\
&+ \delta_{i+1} \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{B}_{i+1}(\mathbf{r}_y,\mathbf{x})\tilde{V}_i(\mathbf{x}) \\
&+ \eta_{i+1} \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \tilde{C}_{i+1}(\mathbf{r}_y,\mathbf{x})\tilde{V}_i(\mathbf{x}) \\
&= \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}}} \left(\gamma_{i+1}\tilde{A}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \delta_{i+1}\tilde{B}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \tilde{C}_{i+1}(\mathbf{r}_y,\mathbf{x})\right)\tilde{V}_i(\mathbf{x})
\end{aligned} \tag{13}$$

Therefore, by computing the bookkeeping table of $\tilde{V}_{i+1}(\mathbf{x})$ together with $\left(\gamma_{i+1}\tilde{A}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \delta_{i+1}\tilde{B}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \tilde{C}_{i+1}(\mathbf{r}_y,\mathbf{x})\right)$ with complexity $O(N_i + N_{i+1})$, $\mathcal{P}$ and $\mathcal{V}$ can run a sumcheck protocol within $O(N_i + N_{i+1})$ proving time. This reduces the correctness to a single evaluation of $\tilde{V}_{i+1}(\mathbf{r}_x)$. Before the next step, we set $\mathbf{r}_{z,i+1} = \mathbf{r}_x$.

After the protocol for all d layers, the correctness is reduced on a single point of the input $\tilde{V}_d(\mathbf{r}_{z,d})$, which $\mathcal{V}$ can compute directly (or access the value through an oracle). Consequently, the total proving time is $O(N_\Sigma)$.

As for the verification complexity, $\mathcal{V}$ needs to verify the intermediate messages received from $\mathcal{P}$ in all stages, which sum up to be $O(d \log N)$ verifying time. In addition, $\mathcal{V}$ needs to compute $\tilde{V}_0(\mathbf{r}_{z,0})$ at the beginning of the scheme in $O(N_0)$,

$\tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{r}_{z,i+1})$, $\tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{r}_{z,i+1})$, and $\tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{r}_{z,i+1})$ for each layer in $O(T)$, and computing $\tilde{V}_d(\mathbf{r}_{z,d})$ at the end in $O(N_d)$. Hence the total verifying complexity is $O(N_0 + N_d + dlogN_\Sigma + T)$.

The soundness error is derived from the soundness error of the sumcheck protocol.

The full interactive proof protocol is in Protocol 2

5 Efficient Interactive Proof Protocol for Data-parallel Boolean Circuit

In this section, we present an efficient interactive proof for a data-parallel Boolean circuit in the form of an LR1CS instance. By data-parallel circuit, we mean a circuit composed of identical sub-circuits that are applied independently to different pieces of data. Data-parallel Boolean circuits have many applications in blockchain and other scenarios, such as computing hashes and digital signatures in applications such as zkRollups, zkEVM, and zkBridge.

To construct our protocol, we first provide the building blocks to efficiently compute the inner product in two different settings. These building blocks are presented in Section 5.1 and 5.2. Then, in Section 5.3, we present the complete interactive proof protocol for the data-parallel Boolean circuit.

5.1 Inner Product between A Field Vector And Bit Vectors

We observe that the inner product between a field vector and several bit vectors is with the same form of multi-scalar multiplication, therefore, we exploit the Pippenger algorithm and achieve a sublinear time algorithm in terms of the field operations. Formally, suppose given a field vector $\mathbf{f} \in \mathbb{F}^\ell$ and q bit vectors $\mathbf{b}_i \in \{0,1\}^\ell$ for $0 \leq i < q$, the goal is to compute $T[i] = \langle \mathbf{f}, \mathbf{b}_i \rangle$. With the Pippenger algorithm, we can split the field vector into w-sized segments $\{\mathbf{s}_k\}_{0 \leq k < \ell/w}$ and compute a table of all inner product between each $\mathbf{s}_k$ and bit strings from $\{0,1\}^w$. Then, to compute the result of the inner product for each vb_i, we can directly look up the answer of each slice in the table, and add them together. The complete algorithm is described in Algorithm 1. The total complexity of this algorithm is $O\left(2^w \frac{\ell}{w} + q\frac{\ell}{w}\right)$ field additions.

5.2 2-D Inner product to Maintain Book-keeping Table

Suppose the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ want to go through a sumcheck protocol to prove the following inner product equation:

$$\sigma(\mathbf{r}_z) = \sum_{\mathbf{y} \in \{0,1\}^n} \tilde{\beta}_n(\mathbf{r}_z, \mathbf{y}) \tilde{f}(\mathbf{y}) \tilde{g}(\mathbf{y}) \tag{14}$$

where $\tilde{f}(\mathbf{y})$ and $\tilde{g}(\mathbf{y})$ are two MLEs two binary vectors $f(\mathbf{y})$ and $g(\mathbf{y})$. Therefore, from Lemma 1 in [29], the proving time is $O(N)$ where $N = 2^n$. Intuitively, in

Protocol 2 (LR1CS-GKR) *The protocol proceeds for the d-layer circuit.*

- *In the first round, $\mathcal{V}$ sends $\mathbf{r}_{z,0} \leftarrow \mathbb{F}^{n_0}$ to $\mathcal{P}$. Both $\mathcal{P}$ and $\mathcal{V}$ computes $\tilde{V}_0(\mathbf{r}_{z,0})$.*
- *In the i-th round, where $0 \leq i \leq d-1$, $\mathcal{P}$ and $\mathcal{V}$ reduce the correctness of $\tilde{V}_i(\mathbf{r}_{z,i})$ to the previous layer in the following two stages.*

  In the first stage, $\mathcal{P}$ and $\mathcal{V}$ prove a rowcheck relation.

  1. *$\mathcal{P}$ initializes three bookkeeping tables $T_f[y]$, $T_g[y]$ and $T_h[y]$ for $y \in \mathbb{Z}_M$, corresponding to the functions:*

$$f(\mathbf{y}) = \sum_{\mathbf{x} \in \{0,1\}^{n_i}} \tilde{A}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x})$$
$$g(\mathbf{y}) = \sum_{\mathbf{x} \in \{0,1\}^{n_i}} \tilde{B}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x})$$
$$h(\mathbf{y}) = \sum_{\mathbf{x} \in \{0,1\}^{n_i}} \tilde{C}_{i+1}(\mathbf{y}, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x})$$

     Then $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol for the following equation:

$$\tilde{V}_i(\mathbf{r}_{z,i}) = \sum_{\mathbf{y} \in \{0,1\}^{n_i}} f(\mathbf{y}) g(\mathbf{y}) + h(\mathbf{y})$$

     During the protocol, $\mathcal{V}$ sends the random challenges $\mathbf{r}_y \leftarrow \mathbb{F}^m$ to $\mathcal{P}$. To prove the last step of the sumcheck protocol, $\mathcal{P}$ sends $f(\mathbf{r}_y)$, $g(\mathbf{r}_y)$, and $h(\mathbf{r}_y)$ to $\mathcal{V}$.

  In the second stage, $\mathcal{P}$ and $\mathcal{V}$ prove a lincheck relation.

  1. *$\mathcal{V}$ sends three random challenges $\gamma_{i+1}, \sigma_{i+1}, \eta_{i+1} \leftarrow \mathbb{F}$ to $\mathcal{P}$.*
  2. *With the randomness received from $\mathcal{V}$, $\mathcal{P}$ initializes two bookkeeping tables $T_{A,B,C}[x]$ and $T_V[x]$ for $x \in \mathbb{Z}_{N_{i+1}}$, corresponding to the functions $f_{A,B,C}(\mathbf{x})$ and $\tilde{V}_{i+1}(\mathbf{x})$, where*

$$f_{A,B,C}(\mathbf{x}) = \gamma_{i+1} \tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \delta_{i+1} \tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \eta_{i+1} \tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{x})$$

     Then $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol for the following equation:

$$\gamma_{i+1} f(\mathbf{r}_y) + \delta_{i+1} g(\mathbf{r}_y) + \eta_{i+1} h(\mathbf{r}_y) = \sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}}} f_{A,B,C}(\mathbf{x}) \cdot \tilde{V}_{i+1}(\mathbf{x})$$

     While running the protocol, $\mathcal{V}$ sends the random challenges $\mathbf{r}_x \leftarrow \mathbb{F}^{n_{i+1}}$ to $\mathcal{P}$.
  3. *$\mathcal{V}$ computes $\tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{r}_x)$, $\tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{r}_x)$ and $\tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{r}_x)$. If $i \leq d-2$, $\mathcal{V}$ receives $\tilde{V}_{i+1}(\mathbf{r}_x)$ from $\mathcal{P}$. Otherwise in the $(d-1)$-th round, $\mathcal{V}$ calculates it from the public input. After that $\mathcal{V}$ verifies the last step of the sumcheck protocol.*
  4. *Before proceeds on the next layer, set $\mathbf{r}_{z,i+1} = \mathbf{r}_x$.*

the i-th round ($0 < \ell < n$), the prover uses dynamic programming to maintain

Algorithm 1 $\{T_0, \ldots, T_{q-1}\} \leftarrow \mathsf{BatchInnerProduct}(\ell, \mathsf{F}, \mathsf{q}, \{\mathsf{B_0}, \ldots, \mathsf{B_{q-1}}\})$

Input: Vector length ℓ, a field vector F and q bit vectors $\{B_i\}_{0 \le i < q}$ and the window size w;
Output: $\{T_0, \ldots, T_{q-1}\}$, where $T_i = \langle F, B_i \rangle$.

for $i = 0, \ldots, \ell/w - 1$ **do** ▷ initialize the lookup table for the field vector
  $T_F[i, 0] = 0$
  $T_F[i, 1] = F[i * w]$
  for $j = 1, \ldots, w - 1$ **do**
    for $k = 0, \ldots, 2^j - 1$ **do**
      $T_F[i, k + 2^j] = T_F[i, k] + F[i * w + j]$
    end for
  end for
end for
for $i = 0, \ldots, q - 1$ **do** ▷ compute each inner product
  $T_i = 0$
  for $j = 0, \ldots, \ell/w - 1$ **do**
    $\mathsf{index} = 0$ ▷ compute the index of the lookup table
    for $k = w - 1, \ldots, 0$ **do**
      $\mathsf{index} = \mathsf{index} * 2 + B_i[j * w + k]$
    end for
    $T[i] = T_i + T_F[j, \mathsf{index}]$
  end for
end for

the three book-keeping tables with $\frac{N}{2^i}$ entries with

$$T_\beta\left[y^{(\ell)}\right] = \tilde{\beta}_{n-i}\left(\mathbf{r}_z, \mathbf{y}[0 : n - \ell] \| \mathbf{r}_y[n - \ell : n]\right)$$
$$T_f\left[y^{(\ell)}\right] = \tilde{f}\left(\mathbf{y}[0 : n - \ell] \| \mathbf{r}_y[n - \ell : n]\right)$$
$$T_g\left[y^{(\ell)}\right] = \tilde{g}\left(\mathbf{y}[0 : n - \ell] \| \mathbf{r}_y[n - \ell : n]\right)$$

where $\mathbf{y}[0 : n - \ell]$ denotes the binary string $(y_0, \ldots, y_{n-\ell-1})$ with the octal form $y^{(\ell)}$, $y^{(\ell)} = \sum_{j=0}^{n-\ell-1} 2^j y_j$. With those tables, it's easy to follow the sumcheck protocol in $O(N)$ field operations. Algorithm 2 and Algorithm 3 describe the process of computing tables and sumcheck protocol correspondingly.

When $\tilde{f}$ and $\tilde{g}$ are MLEs for binary vectors, with corresponding book-keeping table $\mathbf{A}_f^{(\ell)}$ and $\mathbf{A}_g^{(\ell)}$ in ℓ-th round. From Algorithm 2 and Algorithm 3, we observe the two facts:

Algorithm 2 $\mathcal{F} \leftarrow \mathsf{FunctionEvaluations}(f, \mathbf{A}, r_0, \ldots, r_{n-1})$

Input: Multilinear f on ℓ variables, initial bookkeeping table $\mathbf{A}$, random $r_0, \ldots, r_{n-1}$;
Output: All function evaluations $f(b_0, \ldots, b_{n-\ell-2}, t, r_{n-\ell}, \ldots, r_{n-1})$;

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

Algorithm 3 $\{a_0, \ldots, a_{n-1}\} \leftarrow \mathsf{SumCheckProduct}(f, \mathbf{A}_f, g, \mathbf{A}_g, r_0, \ldots, r_{n-1})$

Input: Multilinear f and g, initial bookkeeping tables $\mathbf{A}_f$ and $\mathbf{A}_g$, random $r_0, \ldots, r_n$;
Output: n messages for $a_i(x) = \sum_{b \in \{0,1\}^{n-\ell-1}} f(b, x, r_{n-\ell}, \ldots, r_{n-1}) g(b, x, r_{n-\ell}, \ldots, r_{n-1})$.
Each message a_i consists of 3 elements (a_{i0}, a_{i1}, a_{i2});

1: $\mathcal{F} \leftarrow \mathsf{FunctionEvaluations}(f, \mathbf{A}_f, r_0, \ldots, r_{n-1})$
2: $\mathcal{G} \leftarrow \mathsf{FunctionEvaluations}(g, \mathbf{A}_g, r_0, \ldots, r_{n-1})$
3: **for** $\ell = 0, \ldots, n-1$ **do**
4: **for** $t \in \{0, 1, 2\}$ **do**
5: $a_{it} = \sum_{b \in \{0,1\}^{\ell-i}} f(b, t, r_{n-\ell}, \ldots, r_{n-1}) g(b, t, r_{n-\ell}, \ldots, r_{n-1})$ ▷ All evaluations needed are in $\mathcal{F}$ and $\mathcal{G}$.
6: **end for**
7: **end for**
8: **return** $\{a_0, \ldots, a_{n-1}\}$;

1. The computation process of the book-keeping table is organized as a binary tree from the leaves to the root. Each leaf value corresponds to an entry in the vector, and each intermediate node is a random combination of its two sons. Therefore, $\mathbf{A}_{fg}^{(\ell)}$ is the value of nodes on the ℓ-th layer (leaves are on the 0-th layer).
2. For each round of sumcheck protocol, for each evaluation $t \in \{0,1,2\}$ assigned in the current variable, it essentially computes an "inner product" between all tables.

From the two properties, we observe that for a vector $f[0:n]$ when fixing the block size $w_a = 2^{\log w_a}$, for the first $\log w_a$ stages, there are only w_a essentially different sub-tree defined by the value of $(f(\mathbf{b}\|\mathbf{0}), \ldots, f(\mathbf{b}\|\mathbf{w_a}-\mathbf{1}))$ and w_a^2 different "inner product" defined by the value of $(f(\mathbf{b}\|\mathbf{0}), \ldots, f(\mathbf{b}\|\mathbf{w_a}-\mathbf{1}))$ and $(g(\mathbf{b}\|\mathbf{0}), \ldots, g(\mathbf{b}\|\mathbf{w_a}-\mathbf{1})$ for some $\mathbf{b} \in \{0,1\}^{n-\log w_a}$. Therefore, we further split this stage into two sub-stages. The first sub-stage contains the first $\log w_a$ rounds of the sumcheck protocol. Roughly speaking, we compute a table for all a^2 different cases of "inner product" slices and look up from the table through the value of two vectors before each round. The second stage contains $n - \log w$ rounds, where we fill the book-keeping table and then run a traditional sumcheck protocol.

Sub-stage 1: For the first $\log w_a$ ***rounds*** In the first sub-stage, we use $\mathbf{A}_{fg}^{(\ell)}[\mathbf{v}, \star]$ to defined the ℓ-th layer of the subtree defined by vector $\mathbf{v} \in \{0,1\}^{w_a}$ and randomness $\mathbf{r}_y[n-\ell, n] = r_{y,n-\ell}, \ldots, r_{y,n-1}$. At the beginning of ℓ-th round, we compute all w_a^2 different results of $\mathsf{T}^{(\ell)}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_{w_b}]$ where

$$
\begin{aligned}
&\mathsf{T}^{(\ell)}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_{w_b}, x] \\
&= \sum_{\mathbf{i}' \in \{0,1\}^{\log w_a - \ell - 1}} \left(\sum_{b \in \{0,1\}} \sum_{\mathbf{i} \in \{0,1\}^{\ell}} \mathbf{v}_f(\mathbf{i}'\|\mathbf{i}) \tilde{\beta}_1(b,x) \tilde{\beta}_{\ell}(\mathbf{i}, \mathbf{r}_y[n-\ell, n]) \right) \\
&\cdot \left(\sum_{b \in \{0,1\}} \sum_{\mathbf{i} \in \{0,1\}^{\ell}} \mathbf{v}_g(\mathbf{i}'\|\mathbf{i}) \tilde{\beta}_1(b,x) \tilde{\beta}_{\ell}(\mathbf{i}, \mathbf{r}_y[n-\ell, n]) \right) \\
&\cdot \left(\sum_{b \in \{0,1\}} \sum_{\mathbf{i} \in \{0,1\}^{\ell}} \tilde{\beta}_{\log w_a}(\mathbf{r}_z[\ell - \log w_a : \ell], \mathbf{i}'\|\mathbf{i}) \tilde{\beta}_1(b,x) \tilde{\beta}_{\ell}(\mathbf{i}, \mathbf{r}_y[n-\ell, n]) \right) \\
&\cdot \tilde{\beta}_{\log w_b}(\mathbf{r}_z[n - \log w_a - \log w_b : n - \log w_a], \mathbf{i}_{w_b}) \\
&= \sum_{\mathbf{i}' \in \{0,1\}^{n-\ell-1}} \left(\sum_{b \in \{0,1\}} \tilde{\beta}_1(b,x) \mathbf{A}_{\beta}[\mathbf{i}'\|b] \right) \\
&\cdot \left(\sum_{b \in \{0,1\}} \tilde{\beta}_1(b,x) \mathbf{A}_{fg}^{(\ell)}[\mathbf{v}_f, \mathbf{i}'\|b] \right) \cdot \left(\sum_{b \in \{0,1\}} \tilde{\beta}_1(b,x) \mathbf{A}_{fg}^{(\ell)}[\mathbf{v}_g, \mathbf{i}'\|b] \right) \\
&\cdot \tilde{\beta}_{\log w_b}(\mathbf{r}_z[n - \log w_a - \log w_b : n - \log w_a], \mathbf{i}_{w_b})
\end{aligned}
$$

Then, by enumerating each pair of length-w_a slices in f and g. We can compute the polynomial at the ℓ-th round. Here, the functionality of $\mathbf{i}_{w_b}$ is to avoid too many field multiplications. Then at the ℓ-th round, the polynomial evaluated at $x \in \{0,1,2,3\}$ is computed as follows:

$$p^{(\ell)}(x) = \sum_{i=0}^{\frac{N}{w_a w_b}} \tilde{\beta}_{n-\log w_a - \log w_b}(\mathbf{r}_z[n - \log w_a - \log w_b : n], \mathbf{i}) \cdot$$

$$\left(\sum_{j=0}^{b-1} \mathsf{T}^{(\ell)} \left[f[(iw_b + j)a : (iw_b + j + 1)w_a], g[(iw_b + j)a : (iw_b + j + 1)w_a], \mathbf{j}, x \right] \right)$$

Algorithm 4 describes the process to update book-keeping table and compute $\mathsf{T}^{(\ell)}[\cdot,\cdot,\cdot,\cdot]$. Algorithm 5 describes the complete process of this sub-stage protocol.

The complexity of this sub-stage is $O\left(4^{w_a} w_a w_b + \frac{N}{w_a} \log w_a\right)$ field additions and $O\left(\frac{N}{w_a w_b} \log w_a\right)$ field multiplications.

Sub-stage 2: For the remained $n - \log w_a$ ***rounds*** From the previous sub-stage, we only have a table $\mathbf{A}_{fg}^{(\log w_a - 1)}$ with $\frac{N}{w_a}$ entries, which is the root of all sub-trees. Before the remaining sub-stage, we need to refill the table for $\tilde{f}(\mathbf{r}_z[0 : \log w_a], i_{\log w_a}, \ldots, i_{n-1})$ and $\tilde{g}(\mathbf{r}_z[0 : \log w_a], i_{\log w_a}, \ldots, i_{n-1})$ at each index $(i_{\log w_a}, \ldots, i_{n-1})$. In addition, we compute the book-keeping table for $\tilde{\beta}_{n-\log w_a}(\mathbf{r}_z[\log w_a, n], i_{\log w_a}, \ldots, i_{n-1})$. After that, we proceed with the sumcheck protocol with Algorithm 3.

The complexity of this sub-stage is $O\left(\frac{N}{w_a}\right)$ field additions and multiplications.

5.3 Interactive Proof Protocol

In this section, we present our approach to constructing an efficient proving protocol for data-parallel Boolean circuits. Our strategy involves adjusting the equation that represents the relationship between adjacent layers in a way that yields smaller stages and requires running the sumcheck protocol with only a sublinear number of terms for each layer. To initialize the book-keeping table for the sumcheck protocols, we use Algorithm 1 and Algorithm 4 to accelerate the computation.

Theorem 4. *Let* $\mathbb{x} = (\mathbb{F}, d, N_0, V_d, V_0, \{\bar{A}_i, \bar{B}_i, \bar{C}_i, N_i, K_i\}_{1 \le i \le d}) \in \mathcal{R}_{\mathsf{LR1CS}}$ *for a data-parallel circuit consisting of* $M = 2^m$ *identical sub-circuits. Assume* $N_\Sigma = \sum_{i=0}^{d} N_i$*, there is an interactive proof protocol with soundness* $O(d \log N_\Sigma / |\mathbb{F}|)$*. It uses* $O(d \log N_\Sigma)$ *rounds of interaction and the running time of the prover* $\mathcal{P}$ *is* $O\left(dM + d4^w \log w + \frac{N_\Sigma}{w} \log w\right)$ *field additions and* $O\left(dM + \frac{N_\Sigma}{w} + d4^w \log w\right)$ *field multiplications. Let* T *be the time to evaluate all* $\tilde{A}_i$*,* $\tilde{B}_i$ *and* $\tilde{C}_i$ *at the corresponding random points and* N_d *be the number of outputs, the running time of* $\mathcal{V}$ *is* $O(N_d + d \log N_\Sigma + T)$*.*

Algorithm 4 $\left\{\mathsf{T}^{(\ell)}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x]\right\} \leftarrow \mathsf{ProductBlock}\left(\mathbf{r}_z, \mathbf{A}_{\beta}^{(\ell-1)}, \mathbf{A}_{fg}^{(\ell-1)}, r_{y,n-\ell}\right)$

Table dimensions: $\mathbf{v}_f, \mathbf{v}_g \in \{0,1\}^{w_a}, \mathbf{i}_{w_b} \in \{0,1\}^{\log w_b}, x \in [4]$

Input: The randomness $\mathbf{r}_z$, the table $\mathbf{A}_{fg}^{(\ell)}[\mathbf{v}]$ denoting when the block is $\mathbf{v} \in \{0,1\}^{\log w_a}$, the value of the book-keeping table, the table $\mathbf{A}_{\beta}^{(\ell)}$ the book-keeping table for $\tilde{\beta}_{\log w_a}(\mathbf{r}_z[n - \log w_a : n], \star)$;

Output: $\{\mathsf{T}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x]\}_{\mathbf{v}_f, \mathbf{v}_g \in \{0,1\}^{w_a}, \mathbf{i}_{w_b} \in \{0,1\}^{\log w_b}, x \in [4]}$, denoting the corresponding table entry when proceeding index is $\mathbf{i}_b$, the block vector are $\mathbf{v}_f$ and $\mathbf{v}_g$, evaluated on x.

if $\ell \geq 1$ **then**
  $\mathsf{hsize} \leftarrow 2^{\log w_a - \ell - 1}$
  for $i = 0, \ldots, \mathsf{hsize}$ **do**
    $\mathbf{A}_{\beta}^{(\ell)}[i] \leftarrow \mathbf{A}_{\beta}^{(\ell-1)}[i](1 - r_{y,n-\ell}) + \mathbf{A}_{\beta}^{(\ell-1)}[i + \mathsf{hsize}] r_{y,n-\ell}$;
  end for
  for $v = 0, \ldots, 2^{w_a} - 1$ **do**
    for $i = 0, \ldots, \mathsf{hsize}$ **do**
      $\mathbf{A}_{fg}^{(\ell)}[v, i] \leftarrow \mathbf{A}_{fg}^{(\ell-1)}[v, i](1 - r_{y,n-\ell}) + \mathbf{A}_{fg}^{(\ell-1)}[v, i + \mathsf{hsize}] r_{y,n-\ell}$;
    end for
  end for
else
  Initialize $\mathbf{A}_{\beta}^{(0)}, \mathbf{A}_{fg}^{(0)}$;
end if
for $\mathbf{i}_b \in \{0,1\}^{\log w_b}$ **do**
  for $\mathbf{v}_f = 0, \ldots, 2^{w_a} - 1$ **do**
    for $\mathbf{v}_g = 0, \ldots, 2^{w_a} - 1$ **do**
      for $(x \in \{0, 1, 2, 3\})$ **do**
        $\mathsf{hsize} = 2^{\log w_a - \ell}$
        for $i = 0, \ldots, \mathsf{hsize} - 1$ **do**
          $\beta \leftarrow \mathbf{A}_{\beta}^{(\ell)}[i](1 - x) + \mathbf{A}_{\beta}^{(\ell)}[i + \mathsf{hsize}]$;
          $\sigma_f \leftarrow \mathbf{A}_{fg}^{(\ell)}[\mathbf{f}, i](1 - x) + \mathbf{A}_{fg}^{(\ell)}[\mathbf{f}, i + \mathsf{hsize}]$;
          $\sigma_g \leftarrow \mathbf{A}_{fg}^{(\ell)}[\mathbf{g}, i](1 - x) + \mathbf{A}_{fg}^{(\ell)}[\mathbf{g}, i + \mathsf{hsize}]$;
          $\mathsf{T}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x] \leftarrow \mathsf{T}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x] + \beta \cdot \sigma_f \cdot \sigma_g$
        end for
        $\mathsf{T}^{(\ell)}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x] \leftarrow \mathsf{T}^{(\ell)}[\mathbf{v}_f, \mathbf{v}_g, \mathbf{i}_b, x] \cdot \tilde{\beta}_{\log w_b}(\mathbf{r}_z[n - \log w_a - \log w_b : n - \log w_a], \mathbf{i}_{w_b})$;
      end for
    end for
  end for
end for

Algorithm 5 $\left(p^{(0)}(x), \ldots, p^{(\log w_a - 1)}(x)\right) \leftarrow \mathsf{SumcheckBlockProduct}(\mathbf{r}_z, \mathbf{r}_y, f, g)$

Input: The existing randomness $\mathbf{r}_z$, the randomness from the verifier $\mathbf{r}_y$, $n = \log N$, block size w_a, two vectors f and g with length N;
Output: $p^{(0)}(x), \ldots, p^{(\log w_a - 1)}(x)$, which are the polynomial sent in the first $\log w_a$ rounds.

for $\ell = 0, \ldots, \log w_a - 1$ **do**
  $\mathsf{T}^{(\ell)} \leftarrow \mathsf{ProductBlock}(\mathbf{r}_z, \mathbf{A}_\beta^{(\ell-1)}, \mathbf{A}_{fg}^{(\ell-1)}, r_{y,n-\ell})$
  $p^{(\ell)}(x) \leftarrow 0$ for $x \in \{0, 1, 2, 3\}$;
  for $i = 0, \ldots, \frac{N}{w_a w_b}$ **do**
    $p[i](x) \leftarrow 0$ for $x \in \{0, 1, 2, 3\}$;
    for $j = 0, \ldots, w_b - 1$ **do**
      $\mathbf{f} = f[(iw_b + j)w_a : (iw_b + j + 1)w_a]$;
      $\mathbf{g} = g[(iw_b + j)w_a : (iw_b + j + 1)w_a]$;
      $p[i](x) \leftarrow p[i](x) + \mathsf{T}^{(\ell)}[\mathbf{f}, \mathbf{g}, j, x]$;
    end for
    $p^{(\ell)}(x) \leftarrow p[i](x) * \tilde{\beta}_{n-\log a-\log b}(\mathbf{r}_z[\log(w_a w_b) : n], \mathbf{i})$;
  end for
end for

Algorithm 6 $(\mathbf{A}_\beta, \mathbf{A}_f, \mathbf{A}_g) \leftarrow \mathsf{RefillTable}(\mathbf{A}_\beta^{(\log w_a - 1)}, \mathbf{A}_{fg}^{(\log w_a - 1)}, f, g, \mathbf{r}_z[0 : n - \log w_a], r_{y,n-\log w_a})$

Input: $\mathbf{A}_{fg}$ is the book-keeping table for blocks from the last sub-stage, $\mathbf{r}_z[0 : n - \log w_a]$;
Output: $\mathbf{A}_\beta, \mathbf{A}_f, \mathbf{A}_g$ are book-keeping tables for $\tilde{\beta}_{n-\log a}(\mathbf{r}_z, \mathbf{i}), \tilde{f}(\mathbf{r}_z[0 : n - \log w_a], i), \tilde{g}(\mathbf{r}_z[0 : n - \log w_a], i)$;

Initialize the binary tree from $\mathbf{r}_z[0 : n - \log w_a]$, fill the table $\mathbf{A}_{\beta_z}$ with the leaves;
for $i = 0, \ldots, \frac{N}{w_a} - 1$ **do**
  $\mathbf{A}_\beta[i] \leftarrow \mathbf{A}_\beta[0](1 - r_{y,n-\log w_a}) + \mathbf{A}_\beta[1] r_{y,n-\log w_a}$
  $\mathbf{A}_f[i] = \mathbf{A}_{fg}[f[iw_a : (i+1)w_a], 0](1 - r_{y,n-\log w_a}) + \mathbf{A}_{fg}[f[iw_a : (i+1)w_a], 1] r_{y,n-\log w_a}$;
  $\mathbf{A}_g[i] = \mathbf{A}_{fg}[g[iw_a : (i+1)w_a], 0](1 - r_{y,n-\log w_a}) + \mathbf{A}_{fg}[g[iw_a : (i+1)w_a], 1] r_{y,n-\log w_a}$;
end for

In the theorem above, by definition, $\bar{A}_{i+1}, \bar{B}_{i+1}, \bar{C}_{i+1} \in \mathbb{F}^{N_i \times N_{i+1}}$ to represent the original circuit, while in the following proof, we use A_{i+1}, B_{i+1} and $C_{i+1} \in \mathbb{F}^{N_i/M, N_{i+1}/M}$ to denote the computation of the identical sub-circuit, with $\tilde{A}_{i+1}$, $\tilde{B}_{i+1}$, $\tilde{C}_{i+1}$ as MLE of each matrix. From the equality of the sub-circuits, the number of non-zero entries in A_{i+1}, B_{i+1} and C_{i+1} are $O\left(\frac{N_i}{M}\right)$.

Proof. Instead of proving through the Equation 8, we rewrite the relation formula as follows:

$$\tilde{V}_i(\mathbf{z}\|\mathbf{t}) = \sum_{\mathbf{s}\in\{0,1\}^m} \tilde{\beta}(\mathbf{t},\mathbf{s}) \sum_{\mathbf{y}\in\{0,1\}^{n_i-m}} \tilde{\beta}(\mathbf{z},\mathbf{y}) \cdot \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x}), \tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \right.$$
$$\left. + \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \right) \left(\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \right) \right) \tag{15}$$

Same as the previous section, at the beginning of the protocol, the verifier $\mathcal{V}$ sends a random element $\mathbf{r}_{z,0} \leftarrow \mathbb{F}^{n_0-m}$ and $\mathbf{r}_{t,0} \leftarrow \mathbb{F}^m$ to the prover $\mathcal{P}$. Next, both $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{V}_0(\mathbf{r}_{z,0}\|\mathbf{r}_{t,0})$. After that, for $0 \le i < d$ the two parties go through the following stages:

1. $\mathcal{P}$ **and** $\mathcal{V}$ **prove the row-check relation in terms of** $(\mathbf{y}\|\mathbf{s})$**.** For this stage, the formula to be proved is

$$\tilde{V}_i(\mathbf{r}_{z,i}\|\mathbf{r}_{t,i}) = \sum_{\mathbf{y}\|\mathbf{s}\in\{0,1\}^{n_i}} \tilde{\beta}_m(\mathbf{r}_{t,i},\mathbf{s})\tilde{\beta}_{n_i-m}(\mathbf{r}_{z,i},\mathbf{y}) \cdot (f(\mathbf{y}\|\mathbf{s})g(\mathbf{y}\|\mathbf{s}) + h(\mathbf{y}\|\mathbf{s})) \tag{16}$$

   where

$$f(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \tag{17}$$

$$g(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \tag{18}$$

$$h(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s}) \tag{19}$$

   In this stage, $\mathcal{P}$ and $\mathcal{V}$ first go through Sub-stage 1 by Algorithm 5. After that $\mathcal{P}$ runs Algorithm 6 to initialize the book-keeping table needed in Sub-stage 2. Then $\mathcal{P}$ and $\mathcal{V}$ run Algorithm 3 to finish this stage.
   From the previous conclusion, the first stage cost $O\left(4^{w_a}w_aw_b + \frac{N_{i+1}}{w_a}\log w_a\right)$ field additions and $O\left(4^{w_a}w_aw_b + \frac{N_{i+1}}{w_a}\right)$ multiplications. Before the next stage, $\mathcal{P}$ sends $\mathcal{V}$ three evaluations $f(\mathbf{r}_y\|\mathbf{r}_s)$, $g(\mathbf{r}_y\|\mathbf{r}_s)$ and $h(\mathbf{r}_y\|\mathbf{r}_s)$, with randomness $(\mathbf{r}_y\|\mathbf{r}_s) \leftarrow \mathbb{F}^{n_i}$ from the verifier.
2. $\mathcal{P}$ **and** $\mathcal{V}$ **prove a lincheck relation in terms of x.** After the first stage, the variable $\mathbf{s}$ and $\mathbf{y}$ has been assigned to $\mathbf{r}_s$ and $\mathbf{r}_y$. Before the next phase,

$\mathcal{V}$ also sends $(\gamma_{i+1}, \delta_{i+1}, \eta_{i+1})$ to $\mathcal{P}$. Then the two parties run a sumcheck protocol through the following summation formula:

$$\begin{aligned}
&\gamma_{i+1} f(\mathbf{r}_y \| \mathbf{r}_s) + \sigma_{i+1} g(\mathbf{r}_y \| \mathbf{r}_s) + \eta_{i+1} h(\mathbf{r}_y \| \mathbf{r}_s) \\
&= \gamma_{i+1} \sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}-m}} \tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s) \\
&+ \sigma_{i+1} \sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}-m}} \tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s) \\
&+ \eta_{i+1} \sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{x}) \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s) \\
&= \sum_{\mathbf{x} \in \{0,1\}^{n_{i+1}-m}} (\gamma_{i+1} \tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \sigma_{i+1} \tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \eta_{i+1} \tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{x})) \\
&\qquad \cdot \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s)
\end{aligned} \tag{20}$$

During the initialization, we first compute $(\tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{x})$, $\tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{x})$, and $\tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{x}))$ and then the table of $(\gamma_{i+1}\tilde{A}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \sigma_{i+1}\tilde{B}_{i+1}(\mathbf{r}_y, \mathbf{x}) + \eta_{i+1}$ $\tilde{C}_{i+1}(\mathbf{r}_y, \mathbf{x}))$ can be computed from their combination. For the book-keeping table of $\tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s)$. We claim that

$$\tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s) = \sum_{\mathbf{j} \in \{0,1\}^m} \tilde{\beta}_m(\mathbf{r}_s, \mathbf{j}) \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{j}) \tag{21}$$

which is also consistent with the input form of Algorithm 1. Thus we compute $\tilde{V}(\mathbf{x} \| \mathbf{r}_s)$ for $\mathbf{x} \in \{0,1\}^{n_{i+1}-m}$ in the following steps:

- For $0 \leq j < M$ with the binary representation $\mathbf{j}$, compute $F[j] = \tilde{\beta}(\mathbf{r}_s, \mathbf{j})$, which requires $O(M)$ multiplications.
- For $0 \leq x < N_{i+1}/M$, set $B_x[j] = \tilde{V}_{i+1}(\mathbf{x} \| \mathbf{j})$
- Use Algorithm 1 with the input $(M, F, N_{i+1}/M, \{B_x\}_{0 \leq x < N_{i+1}/M})$ to compute $\tilde{V}_{i+1}(\mathbf{x} \| \mathbf{r}_s)$ for $\mathbf{x} \in \{0,1\}^{n_{i+1}-m}$ The cost is $O\left(\frac{M}{w} 2^w + \frac{N_{i+1}}{w}\right)$ field additions.

Therefore, the complexity for the second stage is $O\left(M + \frac{N_{i+1}}{M}\right)$ field multiplications and $O\left(\frac{M}{w} 2^w + \frac{N_{i+1}}{w}\right)$ field additions.

3. Overall, the complexity for a layer is $O(4^{w_a} w_a w_b + \frac{N_{i+1}}{w_a} \log w_a + \frac{M}{w} 2^w + \frac{N_{i+1}}{w})$ field additions and $O(4^{w_a} w_a w_b + \frac{N_{i+1}}{w_a} + M + \frac{N_{i+1}}{M})$ field multiplications.
4. By choosing $w_a, w_b, w = 0.1 \log N_\Sigma, M = N^{0.1}$, $4^{w_a} w_a w_b$ becomes $\tilde{O}(N^{0.2})$ and $2^w \frac{M}{w} = \tilde{O}(N^{0.2})$ which can be absorbed by the larger term in big-O notation. So the simplified complexity is $O(\frac{N_\Sigma}{\log N_\Sigma} \log \log N_\Sigma)$ field additions and $O(\frac{N_\Sigma}{\log N})$ field multiplications.

After going through the protocols for d layers, the complexity is $O\left(\frac{N_\Sigma}{\log N_\Sigma}\right)$ field multiplications and $O\left(\frac{N_\Sigma}{\log N_\Sigma} \log \log N_\Sigma\right)$ field additions. Since field ad-

dition takes $O(\omega(1))$ RAM operations and field multiplication takes $O(\omega^2(1))$ operations, the final complexity is $O\left(\omega(1)\frac{N_\Sigma}{\log N_\Sigma}\log\log N_\Sigma\right)$ RAM operations.

5.4 Polynomial commitment for GKR

To convert the GKR protocol into a succinct argument, as described in [29, 34], a polynomial commitment needs to be added on top of it. For this purpose, we will use the polynomial commitment developed by Orion and Brakedown [31, 14]. We present their result as follows:

Theorem 5 (Polynomial commitment). *There is a polynomial commitment protocol that, given a multivariate polynomial $f(\cdot)$ with a size of $O(\frac{N}{\log N})$, achieves $O(\frac{N}{\log N})$ field operations in the prover's computation, with a succinct verifier.*

5.5 Putting everything together

Combining the polynomial commitment with our GKR protocol, we obtain a succinct argument protocol, which is formally described as follows:

Theorem 6. *Let $T_{PC}(\cdot)$ be the verification time of the polynomial commitment scheme, and $S_{PC}(\cdot)$ be the proof size of the polynomial commitment. There is a succinct argument protocol for Boolean circuits that satisfies the following properties:*

1. *The Boolean circuit has size $O(N)$ and is composed of $O(\sqrt{N})$ copies of identical sub-circuits.*
2. *The input size of the Boolean circuit is $O(\frac{N}{\log N})$.*
3. *Prover's computation: $O(\omega(1)\frac{N}{\log N}\log\log N)$ RAM operations.*
4. *Verifier's computation: $\mathsf{polylog}(N) + T_{PC}(\frac{N}{\log N})$ field operations.*
5. *Proof size: $O(d\log N) + S_{PC}(\frac{N}{\log N})$ field elements.*

References

1. https://www.alchemy.com/overviews/zkevm
2. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: CRYPTO 2021
3. Bootle, J., Chiesa, A., Groth, J.: Linear-time arguments with sublinear verification from tensor codes. In: Theory of Cryptography (2020)
4. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier. In: EUROCRYPT 2022
5. 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
6. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: EUROCRYPT 2020
7. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT 2020. pp. 769–793 (2020)
8. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-Point Zero Knowledge and Its Applications. In: 2nd Conference on Information-Theoretic Cryptography (ITC 2021)
9. Franzese, N., Katz, J., Lu, S., Ostrovsky, R., Wang, X., Weng, C.: Constant-overhead zero-knowledge for ram programs. CCS '21, ACM
10. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint (2019)
11. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT 2013. pp. 626–645 (2013)
12. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating Computation: Interactive Proofs for Muggles. J. ACM **62**(4), 27:1–27:64 (Sep 2015)
13. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)
14. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and post-quantum snarks for r1cs. ePrint, Paper 2021/1043 (2021)
15. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT 2016. pp. 305–326 (2016)
16. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT 2016 (2016)
17. Gruen, A.: Zk white paper: Efficient zk proofs for keccak — polygon | blog (2022), https://polygon.technology/blog/zk-white-paper-efficient-zk-proofs-for-keccak
18. Holmgren, J., Rothblum, R.D.: Faster sounder succinct arguments and iops. p. 474–503. Springer-Verlag, Berlin, Heidelberg (2022)
19. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Extracting correlations. FOCS '09
20. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: STOC (1992)
21. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic Methods for Interactive Proof Systems. J. ACM **39**(4), 859–868 (Oct 1992)
22. Micali, S.: Computationally sound proofs. SIAM J. Comput. (2000)
23. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P 2013. pp. 238–252 (2013)
24. Pippenger, N.: On the evaluation of powers and related problems. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976). pp. 258–263 (1976)
25. Ron-Zewi, N., Rothblum, R.D.: Proving as fast as computing: Succinct arguments with constant prover overhead. STOC 2022, ACM

26. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: CRYPTO 2020. pp. 704–737. Springer International Publishing (2020)
27. Spielman, D.: Linear-time encodable and decodable error-correcting codes. IEEE Transactions on Information Theory **42**(6), 1723–1731 (1996)
28. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Security & Privacy
29. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO (2019)
30. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. CCS 2022
31. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. CRYPTO (2022)
32. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. CCS '21, ACM
33. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In: CCS (2021)
34. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. IEEE Symposium on Security and Privacy (2020)

Protocol 3 (P-LR1CS-GKR) *The protocol proceeds for the d-layer circuit.*

- *In the first round, $\mathcal{V}$ sends $\mathbf{r}_{z,0} \leftarrow F^{n_0-m}, \mathbf{r}_{t,0} \leftarrow \mathbb{F}^m$ to $\mathcal{P}$. Both $\mathcal{P}$ and $\mathcal{V}$ computes $\tilde{V}_0(\mathbf{r}_{z,0}\|\mathbf{r}_{t,0})$.*
- *In the i-th round, where $0 \leq i \leq d-1$, $\mathcal{P}$ and $\mathcal{V}$ reduce the correctness of $\tilde{V}_i(\mathbf{r}_{z,i}\|\mathbf{r}_{t,i})$ to the previous layer in the following two stages.*
  In the first stage, $\mathcal{P}$ and $\mathcal{V}$ prove a rowcheck relation. $\mathcal{P}$ and $\mathcal{V}$ need to prove the following equation:

$$\tilde{V}_i(\mathbf{r}_{z,i}\|\mathbf{r}_{t,i}) = \sum_{\mathbf{y}\|\mathbf{s}\in\{0,1\}^{n_i}} \tilde{\beta}_m(\mathbf{r}_{t,i},\mathbf{s})\tilde{\beta}_{n_i-m}(\mathbf{r}_{z,i},\mathbf{y})f(\mathbf{y}\|\mathbf{s})g(\mathbf{y}\|\mathbf{s}) + h(\mathbf{y}\|\mathbf{s})$$

  where

$$f(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{A}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s})$$

$$g(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{B}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s})$$

$$h(\mathbf{y}\|\mathbf{s}) = \sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} \tilde{C}_{i+1}(\mathbf{y},\mathbf{x})\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{s})$$

  The protocol split this stage into the following two sub-stages:
  1. *For the first $\log a$ rounds, $\mathcal{P}$ and $\mathcal{V}$ run Algorithm 5.*
  2. *$\mathcal{P}$ calls Algorithm 6 to initialize the book-keeping table for $f(\cdot)$, $g(\cdot)$ and $h(\cdot)$. After that, $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol with respect to Algorithm 3.*
  3. *After the final step of the sumcheck protocol, P sends $\mathcal{V}$ three evaluations $f(\mathbf{r}_y\|\mathbf{r}_s)$, $g(\mathbf{r}_y\|\mathbf{r}_s)$ and $h(\mathbf{r}_y\|\mathbf{r}_s)$, where $(\mathbf{r}_y\|\mathbf{r}_s) \leftarrow \mathbb{F}^{n_i}$ are randomness received from $\mathcal{V}$ during the first stage.*

  In the second stage, $\mathcal{P}$ and $\mathcal{V}$ prove a lincheck relation.
  1. *$\mathcal{V}$ sends three random challenge $\gamma_{i+1}, \sigma_{i+1}, \eta_{i+1} \leftarrow \mathbb{F}$ to $\mathcal{P}$.*
  2. *With the randomness received from $\mathcal{V}$, $\mathcal{P}$ initializes two book-keeping tables $T_{A,B,C}[x]$ for $x \in \mathbb{Z}_{N_{i+1}/M}$, and uses Algorithm 1 again to initializes $T_V[x]$, corresponding to the functions $f_{A,B,C}(\mathbf{x})$ and $\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{r}_s)$ where*

$$f_{A,B,C}(\mathbf{x}) = \gamma_{i+1}\tilde{A}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \sigma_{i+1}\tilde{B}_{i+1}(\mathbf{r}_y,\mathbf{x}) + \eta_{i+1}\tilde{C}_{i+1}(\mathbf{r}_y,\mathbf{x})$$

     Then $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol for the following equation:

$$\begin{aligned}&\gamma_{i+1}f(\mathbf{r}_y\|\mathbf{r}_s) + \sigma_{i+1}g(\mathbf{r}_y\|\mathbf{r}_s) + \eta_{i+1}h(\mathbf{r}_y\|\mathbf{r}_s)\\ =&\sum_{\mathbf{x}\in\{0,1\}^{n_{i+1}-m}} f_{A,B,C}(\mathbf{x})\cdot\tilde{V}_{i+1}(\mathbf{x}\|\mathbf{r}_s)\end{aligned}$$

     While running the protocol, $\mathcal{V}$ sends the random challenges $\mathbf{r}_x \in \mathbb{F}^{n_{i+1}-m}$ to $\mathcal{P}$.
  3. *$\mathcal{V}$ computes $\tilde{A}_{i+1}(\mathbf{r}_y,\mathbf{r}_x)$, $\tilde{B}_{i+1}(\mathbf{r}_y,\mathbf{r}_x)$ and $\tilde{C}_{i+1}(\mathbf{r}_y,\mathbf{r}_x)$. If $i \leq d-2$, $\mathcal{V}$ receives $\tilde{V}_{i+1}(\mathbf{r}_x\|\mathbf{r}_s)$ from $\mathcal{P}$. Otherwise in the $(d-1)$-th round, $\mathcal{V}$ calculates it from the public input. After that $\mathcal{V}$ verifies the last step of the sumcheck protocol.*
  4. *Before the next step, set $\mathbf{r}_{z,i+1} = \mathbf{r}_x$, and $\mathbf{r}_{t,i+1} = \mathbf{r}_s$.*