Distributed-Prover Interactive Proofs

Sourav Das[1(✉)], Rex Fernando[2], Ilan Komargodski[3,4], Elaine Shi[2], and Pratik Soni[5]

[1] UIUC, Champaign, IL, USA
souravd2@illinois.edu
[2] Carnegie Mellon University, Pittsburgh, PA, USA
[3] Hebrew University of Jerusalem, Jerusalem, Israel
ilank@cs.huji.ac.il
[4] NTT Research, Sunnyvale, CA, USA
[5] University of Utah, Salt Lake City, UT, USA
psoni@cs.utah.edu

Abstract. Interactive proof systems enable a verifier with limited resources to decide an intractable language (or compute a hard function) by communicating with a powerful but untrusted prover. Such systems guarantee soundness: the prover can only convince the verifier of true statements. This is a central notion in computer science with far-reaching implications. One key drawback of the classical model is that the data on which the prover operates must be held by a single machine.

In this work, we initiate the study of distributed-prover interactive proofs (dpIPs): an untrusted cluster of machines, acting as a distributed prover, interacts with a single verifier. The machines in the cluster jointly store and operate on a massive data-set that no single machine can store. The goal is for the machines in the cluster to convince the verifier of the validity of some statement about its data-set. We formalize the communication and space constraints via the massively parallel computation (MPC) model, a widely accepted analytical framework capturing the computational power of massive data-centers.

Our main result is a compiler that generically augments any verification algorithm in the MPC model with a (computational) soundness guarantee. Concretely, for any language L for which there is an MPC algorithm verifying whether $x \in L$, we design a new MPC protocol capable of convincing a verifier of the validity of $x \in L$ and where if $x \notin L$, the verifier rejects with overwhelming probability. The new protocol requires only slightly more rounds, i.e., a $\mathsf{poly}(\log N)$ blowup, and a slightly bigger memory per machine, i.e., $\mathsf{poly}(\lambda)$ blowup, where N is the total size of the dataset and λ is a security parameter independent of N.

En route, we introduce distributed-prover interactive oracle proofs (dpIOPs), a natural adaptation of the (by now classical) IOP model to the distributed prover setting. We design a dpIOP for verification algorithms in the MPC model and then translate them to "plain model" dpIPs via an adaptation of existing polynomial commitment schemes into the distributed prover setting.

P. Soni—Work was done partially when the author was visiting Carnegie Mellon University.

© International Association for Cryptologic Research 2023
G. Rothblum and H. Wee (Eds.): TCC 2023, LNCS 14369, pp. 91–120, 2023.
https://doi.org/10.1007/978-3-031-48615-9_4

1 Introduction

Interactive proofs are a natural extension of non-determinism and have become a fundamental concept in complexity theory and cryptography. The study of interactive proofs has led to many of the exciting notions that are at the heart of several areas of theoretical computer science, including zero-knowledge proofs [40,41] and probabilistically checkable proofs (PCPs) [4,10,11].

An interactive proof is a protocol between a randomized verifier and a powerful but untrusted prover. The goal of the prover is to convince the verifier regarding the validity of some statement. If the statement is indeed correct, we require that the verifier should accept an honestly generated proof with high probability. Otherwise, if the statement is false, the verifier should reject with high probability any maliciously crafted proof. A particularly interesting and practical case is when the verifier is significantly weaker than the prover in some aspect. Typically, verifiers that are weaker in terms of computational abilities are studied, but other sorts of limitations are relevant.

The standard model of interactive proofs, as described above, has a key limitation: *The data must be held by a prover modeled as a single machine.* A scenario where the data is distributed among multiple parties is not natively supported. Indeed, large organizations nowadays store vast amounts of data, often reaching petabytes or even exabytes in size. To store and efficiently manage such enormous volumes of data, these organizations utilize massive data-centers. With existing succinct arguments, if such an organization takes up the role of the prover, the only way to use existing interactive proofs technology is by essentially aggregating the data at a single machine. However, the latter is physically impossible as there is no one machine that can store so much data.

Motivated by the above scenario, in this work, we study *interactive proofs for distributed provers.* We first define a concrete model that captures the constraints of such a distributed setting, and then design new interactive proofs in our model.

The Distributed Computation Model. We imagine an enormous data-set, the size of which is denoted N. The data is stored in a cluster split among M machines; i.e., every machine stores roughly a size N/M portion of the data-set. As an example, imagine that $N = 10^{17}$ bytes (100 petabytes) and that each machine has a hard-disc capable of storing 10^{13} bytes (10 terabytes). Then, a cluster consisting of $10^4 = 10,000$ machines is needed. (Clearly, there is no single machine capable of storing 100 petabytes).

The distributed prover is the above cluster, consists of M server machines. The verifier is another machine, as powerful as a single machine in the cluster, i.e., it can store N/M bits of information. The goal of the distributed prover is to convince the verifier of the validity of some statement about its data-set. The distributed prover can perform arbitrary communication (server-to-server or server-to-client) and local computation, as long as it respects the space constraint of each machine. If we care about computational complexity of the (honest or malicious) prover, we shall also require that the local computation of the (honest or malicious) provers is polynomial time. Each server machine and the client have their own private source of randomness.

The above logic coincides with the rationale behind the *Massively Parallel Computation* (MPC) model. This model was invented to capture popular modern parallel computation programming paradigms such as MapReduce, Hadoop, and Spark, designed to utilize parallel computation power to manipulate and analyze huge amounts of data. In this model, first introduced by Karloff, Suri, and Vassilvitskii [42], the size N data-set is stored in a distributed manner among M machines. The machines are connected via pairwise communication channels and each machine can only store $S = N^{\epsilon}$ bits of information locally for some $\epsilon \in (0,1)$. Naturally, we assume that $M \geq N^{1-\epsilon}$ so that all machines can jointly at least store the entire data-set.

The primary metric for the complexity of algorithms in this model is their *round complexity*. Reasonable polynomial-time computations that are performed within a machine are considered "for free" since the communication is often the bottleneck. We typically want algorithms in the MPC framework to have a small number of rounds, say, poly-logarithmically or even sub-logarithmically many rounds (in the total data size N). With the goal of designing efficient algorithms in the MPC model, there is an immensely rich algorithmic literature suggesting various non-trivial efficient algorithms for tasks of interest, including graph problems [1–3,6–9,12,15,16,26,28,31], clustering [13,14,33,39] and submodular function optimization [32,34,46].

Succinct Arguments in the MPC Model. In this work, we study the question of constructing interactive argument systems in the MPC model, where the "prover" is a cluster of machines, each with N^{ϵ} maximum storage, where N is the size of the witness, and the client is also a machine with the same storage restriction. Note that it is unrealistic to achieve an argument system for all polynomial-time computable functions in this model, because there are various results showing that not all such functions can be computed in the MPC model [30,54]. Thus, we aim for the best-possible goal: to prove a statement whose verification algorithm is itself an MPC algorithm.

We design an argument system that supports clusters acting as provers and where the protocol respects the requirements of the MPC model. Specifically, we prove the following theorem.

Theorem 1 (Main result; Informal). *Let $R = \{(x, w)\}$ be any relation which has a massively-parallel verification algorithm Π among $M = N^{1-\epsilon}$ parties each with space N^{ϵ}, where $N = |w|$, and $|x| \leq N^{\epsilon}$.*

Then there exists an argument system Π' for R in the MPC model, which has M space-bounded provers $P_1, \ldots, P_M$, and convinces a space-bounded verifier V that $x \in L_R$. The protocol Π' has space overhead multiplicative in $\mathsf{poly}(\lambda)$ *relative to Π, where λ is a security parameter, and has round overhead multiplicative in* $\mathsf{polylog}(N)$.

Under standard falsifiable cryptographic assumptions, the argument Π' is sound in the CRS model against malicious provers with arbitrary $\mathsf{poly}(N, \lambda)$ *running time and space.*

Our protocol's soundness relies on the existence of groups of hidden order, which can be instantiated based on the RSA assumption [53] or on class groups [27,57].

To put the above result in better context, we mention a recent work of Fernando et al. [35] (building on [36]) who built a secure computation compiler for arbitrary MPC protocols. That is, they compile any MPC protocol into secure counterparts, which still respect the constraints of the model. In particular, their protocol can be used as an argument system in the cluster-verifier model we introduce above. Unfortunately, their compiler relies on (publicly verifiable) succinct non-interactive proofs of knowledge (SNARKs), which are well-known not to be constructible based on falsifiable assumptions [38,49]. Our main contribution, and the main technical challenge we overcome, is achieving such an argument system relying only on falsifiable assumptions. As a bonus, we mention that if we instantiate the hidden order group using class groups, our protocol requires only a common *random* string, whereas the SNARK based solution requires a structured common *reference* string.

1.1 Techniques: Distributed IOPs and Distributed Streaming Polynomial Commitments

To achieve our main result, we use recent work on interactive oracle proofs (IOPs). Recall that the IOP model is a proof system model that combines features of interactive proofs (IPs) and probabilistically checkable proofs (PCPs). In this model, the verifier is not required to read the prover's messages in their entirety; rather, the verifier has oracle access to some of the prover's messages (viewed as strings), and may probabilistically query these messages during its interaction with the prover. IOPs strictly generalize PCPs, and serve as a convenient intermediate model for getting succinct "plain model" protocols. Many recent succinct arguments have been constructed by first giving a protocol in the IOP model, and then using a vector commitment or polynomial commitment to instantiate the IOP oracle [18,21,22,27,29,37].

We extend the IOP model to a setting where the prover is distributed — here on referred to as the distributed IOP. We imagine a prover that is made up of a collection of servers that can communicate between themselves via peer-to-peer channels, as in the classical distributed cluster-verifier MPC model. But, communication between any server and the verifier occurs as in the IOP model: the verifier has oracle access to a large string committed to by the server, in addition to being able to communicate directly with any of the parties comprising the server.

We build a distributed IOP in the MPC model analogous to the "plain model" protocol we stated above. Specifically, given a distributed, massively-parallel protocol Π for verifying a relation R, we construct a distributed argument system Π' which works in this new IOP model, and where a distributed group of provers convince a verifier V that some $x \in L_R$. Our argument uses a *polynomial commitment oracle,* where each prover first streams evaluations of some multilinear polynomial W over some subset of the Boolean hypercube, and where at the end the provers have collectively defined W by their evaluations. The verifier then interacts with the prover and queries this polynomial IOP oracle in order to verify the statement x.

Our IOP is inspired by the work of Blumberg et al. [23], who give an IOP for RAM programs, where the prover's running time and space are approximately preserved in relation to the running time and space of the verification algorithm. At a very high level, the [23] IOP has the prover commit to a polynomial $\hat{W}$, which encodes the RAM computation, and then has the prover and verifier run a sumcheck argument in relation to a polynomial h that is based on $\hat{W}$. The polynomial h has the property that it can be evaluated at any point via a constant number of evaluations of $\hat{W}$. At the end of the sumcheck, the verifier can thus query the IOP oracle in order to do the final random evaluation of h.

We would like to use a similar strategy to [23], having the provers encode a polynomial $\hat{W}$ which encodes the MPC computation, and then using a sumcheck argument to verify the truthfulness of $\hat{W}$. However, since $\hat{W}$ now encodes an *interactive protocol between RAM programs* Π_L, instead of just a RAM computation, it is unclear how the provers would be able to generate sumcheck messages without rerunning the MPC protocol many times, thus blowing up the communication complexity.

To solve this, we use several ideas. First, for each round of the MPC protocol, the provers commit to a concatenation π_r of their states after the round is finished, using a Merkle tree-based succinct commitment. This defines a statement (r, π_{r-1}, π_r), where a witness for this statement is a set of decommitments for π_{r-1} and π_r which show honest behavior during this round. If we can build a knowledge-sound argument for this statement which works in the MPC model and is round-efficient, this is sufficient to build an argument for honest execution of the whole protocol Π_L. We then design an IOP similar to [23] for proving the statement (r, π_{r-1}, π_r). Note that even though we have reduced to proving honesty of one round, we still have the problem that knowledge of $\hat{W}$ is spread across all the provers, and no single prover knows the whole description of $\hat{W}$. Thus it is still unclear how the provers will generate the sumcheck provers' messages in a round-efficient way. The main technical part of our paper deals with how to do this.

Polynomial Commitments. Once we have an IOP for L, we still need to instantiate it using a polynomial commitment scheme. Informally, a polynomial commitment scheme allows a prover to commit to some low degree polynomial f, and provide evaluations $f(x)$ to a verifier along with a (interactive) proof that the provided evaluation is consistent with the commitment. Polynomial commitments were introduced by [43] and have recently drawn significant attention due to their use in compiling oracle proof systems (e.g., PCPs and IOPs) into real world proof systems (e.g., arguments). A sequence of works [5,17,19,24,25,27,44,47,52,55,56,59] have studied several different aspects of efficiency including getting constant-sized proofs/commitments, sublinear (even polylogarithmic) time verification, as well as linear prover time. However, these works consider a monolithic prover that stores the entire polynomial locally. This is in stark contrast with our setting where there are multiple provers $P_1, \ldots, P_M$, each of which only have *streaming* access to a *small piece* of the description of the polynomial. Looking ahead, the polynomial in our context is the description of the transcript of the RAM computation, which can be generated as a stream.

The works that come closest to our requirements are that of Block et al. [21,22] who introduced the *streaming model* of access where a monolithic prover has streaming access to the description of the polynomial. They build a logarithmic round polynomial commitment scheme in the streaming model where the prover's memory usage is logarithmic, the prover time is quasilinear, and requires only a logarithmic number of passes over the stream. Using such a polynomial commitment scheme they build a succinct argument for RAM computation where the prover is both time- and space- optimal. The key structural property of their construction that allows for this small-space implementation in the streaming model is: they show that *for each of the logarithmic rounds*, prover's messages in the interactive proof of consistency can be expressed as a *linear combination of the elements in the description stream.* Therefore, it is sufficient for the monolithic prover to take a single pass over its stream to compute its message in every round. Although, their work still considers a monolithic prover, this structural property is the starting point of our work. In particular, we observe that the natural adaptation of Block et al. [22] commitment scheme to our setting suffices for our purposes. In fact, when the cluster of provers $P_1, \ldots, P_M$ is viewed as a monolithic prover, then the two schemes are identical. This allows us to base our security on that of Block et al. [22], which in turn, is based on groups of hidden order (e.g., RSA and class groups). Due to the above structural property, in each of the rounds, each of the provers in the cluster can (a) first compute their contributions to this round's message in small space, while making a single pass over their stream, and (b) then all provers can combine their contributions in logarithmic (in M) rounds via a tree-based protocol to compute the full round message.

We present our construction in the MPC model in Sect. 6.2. Along the way, we introduce the definition of polynomial commitments in the MPC model tailored to the case of multilinear polynomials in Sect. 4.

1.2 Related Work

The terminology of distributed interactive proofs appeared in several prior works, all of which differ significantly from our notion. The works [20,45,50] all study a variant of interactive proofs where the verifier is distributed but the prover is a single machine. The work of [51,58] allow multiple (potentially mutually-distrusting) provers to efficiently derive a single SNARK for a large statement/witness pair. While their goal on the surface is similar to ours, both works inherently require non-falsifiable assumptions since they rely on SNARKs. In contrast, the main contribution of our work is in building a succinct argument system that *does not* require non-falsifiable assumptions.

1.3 Organization

The rest of the paper is organized as follows. Section 2 contains preliminaries. In Sect. 3, we define the MPC model and security properties required for argument systems in this model. In Sect. 4, we define polynomial commitments that work with distributed committers. Section 5 contains the main construction of succinct arguments in the MPC model. Section 6 contains our adaptation of the [22] polynomial commitment.

2 Preliminaries

Let S be some finite, non-empty set. By $x \leftarrow S$ we denote the process of sampling a random element x from S. For any $k \in \mathbb{N}$, by S^k we denote the set of all sequences/vectors of length k containing elements of S where $S^0 = \{\epsilon\}$ for empty string ϵ. We let $\mathbb{F} = \mathbb{F}_p$ denote a finite field of prime cardinality p. We assume that $\vec{b} = (b_n, \ldots, b_1)$, where b_n is the most significant bit and b_1 is the least significant bit. For bitstrings $\vec{b} \in \{0,1\}^n$, we naturally associate $\vec{b}$ with integers in the set $\{0, \ldots, 2^n - 1\}$, i.e., $\vec{b} = \sum_{i=1}^{n} b_i \cdot 2^{i-1}$. For any two equal sized vectors $\vec{u}, \vec{v}$, by $\vec{u} \odot \vec{v}$ we denote the coordinate-wise multiplication of $\vec{u}$ and $\vec{v}$. We use uppercase letters to denote matrices, e.g., $A \in \mathbb{Z}^{m \times n}$. For $m \times n$ dimensional matrix A, $A(i, *)$ and $A(j, *)$ denote the i-th row and j-th column of A, respectively.

Notation for Matrix-Vector "Exponents". For some group $\mathbb{G}$, $A \in \mathbb{Z}^{m \times n}$. $\vec{u} = (u_1, \ldots, u_m) \in \mathbb{G}^{1 \times m}$, and $\vec{v} = (v_1, \ldots, v_m)^\top \in \mathbb{G}^{n \times 1}$, we let $\vec{u} \star A$ and $A \star \vec{v}$ denote a matrix-vector exponent, defined for every $j \in [n]$, $i' \in [m]$ as

$$(\vec{u} \star A)_j = \prod_{i=1}^{m} u_i^{A(i,j)} \ ; \ (A \star \vec{v})_{i'} = \prod_{j'=1}^{n} v_{j'}^{A(i',j')} \ ,$$

For any vector $\vec{x} \in \mathbb{Z}^n$ and group element $g \in \mathbb{G}$, we define $g^{\vec{x}} = (g^{x_1}, \ldots, g^{x_n})$. Finally, for $k \in \mathbb{Z}$ and a vector $\vec{u} \in \mathbb{G}^n$, we let $\vec{u}^k$ denote the vector $(u_1^k, \ldots, u_n^k)$.

2.1 Multilinear Polynomials

An n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ is *multilinear* if the individual degree of each variable in f is at most 1.

Fact 1. *A multilinear polynomial $f : \mathbb{F}^n \to \mathbb{F}$ (over a finite field $\mathbb{F}$) is uniquely defined by its evaluations over the Boolean hypercube. Moreover, for every $\vec{\zeta} \in \mathbb{F}^n$,*

$$f(\vec{\zeta}) = \sum_{\vec{b} \in \{0,1\}^n} f(\vec{b}) \cdot \prod_{i=1}^{n} \chi(b_i, \zeta_i) \ ,$$

where $\chi(b, \zeta) = b \cdot \zeta + (1 - b) \cdot (1 - \zeta)$.

As a shorthand, we will often denote $\prod_{i=1}^{n} \chi(b_i, \zeta_i)$ by $\overline{\chi}(\vec{b}, \vec{\zeta})$ for $n = |\vec{b}| = |\vec{\zeta}|$.

Notation for Multilinear Polynomials. Throughout, we denote a multilinear polynomial f by the 2^n sized sequence $\mathcal{Y}$ containing its evaluations over the Boolean hypercube. That is, $\mathcal{Y} = (f(\vec{b}) : \vec{b} \in \{0,1\}^n)$, and denote the evaluation of the multilinear polynomial defined by $\mathcal{Y}$ on the point $\vec{\zeta}$ as $\mathsf{ML}(\mathcal{Y}, \vec{\zeta}) = \sum_{\vec{b} \in \{0,1\}^n} \mathcal{Y}_{\vec{b}} \cdot \overline{\chi}(\vec{b}, \vec{\zeta})$.

3 Model Definition

In the massively-parallel computation (MPC) model, there are M parties (also called machines) and each party has a local space of S bits. The input is assumed to be distributed across the parties. Let N denote the total input size in bits; it is standard to assume $M \geq N^{1-\varepsilon}$ and $S = N^{\varepsilon}$ for some small constant $\varepsilon \in (0, 1)$. Note that the total space is $M \cdot S$ which is large enough to store the input (since $M \cdot S \geq N$), but at the same time it is not desirable to waste space and so it is commonly further assumed that $M \cdot S \in \tilde{O}(N)$ or $M \cdot S = N^{1+\theta}$ for some small constant $\theta \in (0, 1)$. Further, assume that $S = \Omega(\log M)$.

At the beginning of a protocol, each party receives an input, and the protocol proceeds in rounds. During each round, each party performs some local computation given its current state (modeled as a RAM program with maximum space S), and afterwards may send messages to some other parties through private authenticated pairwise channels. An MPC protocol must respect the space restriction throughout its execution, even during the communication phase—to send a message at the end of a round, a party must write that message in some designated place in memory, and in order to receive a message at the end of a round, a party must reserve some space in memory equal to the size of the message. This in turn implies that each party can send or receive at most S bits in each round. An MPC algorithm may be randomized, in which case every machine has a sequential-access random tape and can read random coins from the random tape. The size of this random tape is not charged to the machine's space consumption.

3.1 Succinct Arguments in the MPC Model

We are interested in building a succinct argument in this model for some NP language L, where the witness $w = (w_1, \ldots, w_M)$ for $x \in L$ has size much larger than S. The prover role is carried out by a group of S-space-bounded parties $P_1, \ldots, P_M$, each of which has the statement x and one piece w_i of the witness. They work together to convince a verifier V, which is also S-space-bounded. Since any prover must at least be powerful enough to verify that $(x, w) \in R_L$, and the MPC model is not known to capture P when the rounds are bounded, we only consider languages L where the verification algorithm $R_L \colon ((x, w_1), \ldots, (x, w_M)) \to \{0, 1\}$ is implementable by a MPC protocol Π_L where each party is S-space-bounded. Given such a protocol, our goal is to build a new MPC protocol Π'_L between $M+1$ parties $P_1, \dots, P_M, V$, where P_i has input (x, w_i) and V has input x, which satisfies the properties discussed below.

Communication Model and Setup. We assume a synchronous setting, with pairwise channels between parties. We also allow for a CRS $\mathsf{Setup}(1^\lambda) \rightarrow (\alpha_1, \dots, \alpha_M)$, where party i receives α_i at the beginning of the protocol. Since each party must store some α_i, it is clear that $|\alpha_i| \leq S$ for all i. Looking ahead, in our protocol, all parties get the same CRS string α which is a description of a group of size 2^λ, that is, $\alpha_i = \alpha$ for all $i \in [M]$.

Efficiency Requirements. We want to build a protocol Π'_L which has efficiency properties as close as possible to the original verification protocol Π_L. Specifically, if in Π_L each party uses space bounded by S, in Π'_L each party's space should be bounded by $S \cdot p(\lambda)$, for some fixed polynomial p. Moreover, if Π_L takes r rounds, Π'_L should take a small multiplicative factor $r \cdot \beta$ rounds. In this paper, we set $\beta = \mathsf{polylog}(N)$.

Security Requirements. Let α be the output of the setup algorithm, and denote with

$$\Pi'_L \langle [P_1, \dots, P_M], V \rangle \left(1^\lambda, \alpha, x, w = (w_1, \dots, w_M)\right)$$

the output of the protocol Π'_L with interactive RAM programs $P_1, \dots, P_M$ playing the roles of the M provers, and with the interactive RAM program V playing the role of the verifier, where each P_i is initialized with input $(1^\lambda, \alpha, x, w_i)$, and V is initialized with input $(1^\lambda, \alpha, x)$. Similarly, denote with

$$\Pi'_L \langle \mathcal{A}, V \rangle \left(1^\lambda, \alpha, x, w = (w_1, \dots, w_M)\right)$$

the output of the protocol Π'_L with an interactive *monolithic* RAM program $\mathcal{A}$ playing the role of all provers $P_1, \dots, P_M$, and with the interactive RAM program V playing the role of the verifier, where $\mathcal{A}$ is initialized with the inputs of all P_i as defined above, and V is initialized in the same way as above.

We require Π'_L satisfies completeness and soundness, defined as follows.

Definition 1 (Completeness). *Let L be a language with a corresponding MPC protocol Π_L which implements the verification functionality for R_L. For all $(x, w) \in R_L$ and for all λ, letting $m = m(|x|)$,*

$$\Pr\left[\Pi'_L \langle [P_1, \dots, P_M], V \rangle \left(1^\lambda, \mathsf{Setup}(1^\lambda), x, w\right) = 1\right] = 1,$$

where $P_1, \dots, P_M$ (resp., V) are the honest provers (resp., verifier), and the probability is taken over random coins of the parties and of the setup algorithm.

Definition 2 (Soundness). *Let L be a language with a corresponding MPC protocol Π_L which implements the verification functionality R_L. Fix a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ takes as input the security parameter and the output of* Setup*, and chooses an input x, and where $\mathcal{A}_2$ plays the roles of* *the provers $P_1, \ldots, P_M$. Then Π_L is said to satisfy soundness if there exists a negligible function* negl *such that for all λ,*

$$\Pr\left[\begin{array}{c} x \notin L \wedge \\ \Pi'_L \langle \mathcal{A}_2, V\rangle \left(1^\lambda, \alpha, x, \perp\right) = 1 \end{array} : \begin{array}{l} (\alpha_1, \ldots, \alpha_M) \leftarrow \mathsf{Setup}(1^\lambda) \\ x \leftarrow \mathcal{A}_1(\lambda, \alpha_1, \ldots, \alpha_M) \end{array}\right] < \mathsf{negl}(\lambda).$$

To prove soundness of our protocol, we show the stronger property of witness-extended emulation as formalized by Lindell [48]. Intuitively, witness-extended emulation requires the existence of an efficient extractor that can simulate an adversarial prover's view while extracting the underlying witness. Below we formally extend the standard definition to the MPC setting in the natural way.

Definition 3 (Witness-Extended Emulation). *Let L be a language with a corresponding MPC protocol Π_L which implements the verification functionality R_L. Fix a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ takes as input the security parameter and the output of* Setup *and chooses an input x along with a private state σ, and where $\mathcal{A}_2$ takes this σ as input and plays the roles of the provers $P_1, \ldots, P_M$. Then Π_L is said to satisfy witness-extended emulation with respect to L (and R_L) if there exists an (expected) PPT machine $\mathcal{E}$ (called the "extractor") and a negligible function* negl *such that the following holds. Define two distributions $\mathcal{D}_1^\lambda$ and $\mathcal{D}_2^\lambda$ based on $\mathcal{A}$ and $\mathcal{E}$, as follows:*

- *$\mathcal{D}_1^\lambda$: Compute the setup $\alpha \leftarrow \mathsf{Setup}(1^\lambda)$ and then compute $(x, \sigma) \leftarrow \mathcal{A}_1(1^\lambda, \alpha)$, then output $(\alpha, r_\mathcal{A}, r_V, x, \tau)$, where τ is the transcript of messages obtained by the execution $\Pi'_L \langle \mathcal{A}_2(\sigma), V\rangle \left(1^\lambda, \alpha, x, \perp\right)$, $r_\mathcal{A}$ is the random tape of $\mathcal{A}_1$ and $\mathcal{A}_2$, and r_V is the random tape of V.*
- *$\mathcal{D}_2^\lambda$: Compute the setup $\alpha \leftarrow \mathsf{Setup}(1^\lambda)$ and then compute $(x, \sigma) \leftarrow \mathcal{A}_1(1^\lambda, \alpha)$, then output $(\alpha, r_\mathcal{A}, r_V, x, \tau, w) \leftarrow \mathcal{E}^\mathcal{O}(1^\lambda, \alpha, x)$, where $\mathcal{O}$ is an oracle which provides an execution of $\Pi'_L \langle \mathcal{A}_2(\sigma), V\rangle \left(1^\lambda, \alpha, x, \perp\right)$, and allows for rewinding of the protocol and choosing the randomness of $\mathcal{A}_2$ during each round.*

With respect to these distributions, for all λ, the following holds:

1. *The distributions $\mathcal{D}_1^\lambda$ and $\mathcal{D}_2^\lambda\big|_{\alpha,r,x,\tau}$ are identical, where $\mathcal{D}_2^\lambda\big|_{\alpha,r,x,\tau}$ is the restriction of $\mathcal{D}_2$ to the first four components of the tuple (α, r, x, τ, w).*
2. *It holds that $\Pr\left[V \text{ accepts and } (x, w) \notin R_L : (\alpha, r, x, \tau, w) \leftarrow \mathcal{D}_2^\lambda\right] < \mathsf{negl}(\lambda)$.*

4 Defining Multilinear Polynomial Commitments in the MPC Model

In this section, we discuss how to define a polynomial commitment scheme which works in the MPC model starting with a discussion on how the polynomial is distributed across all of the M many S-space-bounded parties. Let M be a power of 2 and let $\mathcal{Y} \in \mathbb{F}^N$ define an n variate multilinear polynomial where $N = 2^n$. We assume that $\mathcal{Y}$ is distributed across all parties in the following way: Let $\{I_1, \ldots, I_M\}$ be the canonical partition of $\{0,1\}^n$, that is, $I_i = \{(i-1) \cdot N/M, \ldots, i \cdot N/M - 1\}$. We associate each party P_i with the subset I_i, and assume that P_i holds only the partial vector $\mathcal{Y}_i$ containing elements from $\mathcal{Y}$ restricted to the indices in I_i. That is,

$$\mathcal{Y}_i = (\mathcal{Y}_{\vec{b}})_{\vec{b} \in I_i} \ .$$

Furthermore, for the canonical partition, if i-th party holds the partial vector $\mathcal{Y}_i$, then they collectively define the multilinear polynomial $\mathcal{Y}$ where $\mathcal{Y} = \mathcal{Y}_1 \mid\mid \mathcal{Y}_2 \mid\mid \ldots \mid\mid \mathcal{Y}_M$, where $\mid\mid$ refers to the concatenation of two vectors.

Definition 4 (Multilinear Polynomial Commitment Syntax). *A multilinear polynomial commitment has the following syntax.*

- $\mathsf{PC.Setup}(1^\lambda, p, 1^n, M) \rightarrow pp$*: On input the security parameter* 1^λ *(in unary), a field size p less than* 2^λ*, the number of variables* 1^n *(also in unary), and the number of parties M, the setup algorithm* $\mathsf{PC.Setup}$ *is a randomized PPT algorithm that outputs a CRS pp whose size is at most* $\mathsf{poly}(\lambda, n, \log(M))$*.*
- $\mathsf{PC.PartialCom}(pp, \mathcal{Y}_i) \rightarrow (\mathsf{com}_i; \mathcal{Z}_i)$*: On input a CRS pp, and a vector* $\mathcal{Y}_i \in \mathbb{F}$ *which is the description of a multilinear polynomial restricted to the set* $I_i \subset \{0,1\}^n$*,* $\mathsf{PC.PartialCom}$ *outputs a "partial commitment"* com_i *as well as the corresponding decommitment* $\mathcal{Z}_i \in \mathbb{Z}$*.*
- $\mathsf{PC.CombineCom}(pp, \{\mathsf{com}_i\}_{i \in [M]}) \rightarrow \mathsf{com}$*: This is an interactive PPT protocol in the MPC model computing the following functionality: each party* P_i *holds the string* (pp, com_i)*, they jointly compute the full commitment* com *such that* P_1 *learns* com*, and outputs it.*
- $\mathsf{PC.PartialEval}(pp, \mathcal{Y}_i, \vec{\zeta}) \rightarrow y_i$*: On input a CRS pp, a partial description vector* $\mathcal{Y}_i$*, and an evaluation point* $\vec{\zeta} \in \mathbb{F}^n$*,* $\mathsf{PC.PartialEval}$ *is a PPT algorithm that outputs the partial evaluation* y_i*.*
- $\mathsf{PC.CombineEval}(pp, \{y_i\}_{i \in [M]}, \vec{\zeta}) \rightarrow y$*: This is an interactive PPT protocol in the MPC model computing the following functionality: each party* P_i *holds the string* (pp, y_i)*, they jointly compute the full evaluation y such that* P_1 *learns y, and outputs it.*
- $\mathsf{PC.IsValid}(pp, \mathsf{com}, \mathcal{Y}, \mathcal{Z}) \rightarrow 0$ *or* 1*: On input the CRS pp, a commitment* com*, a multilinear polynomial* $\mathcal{Y}$ *and a decommitment* $\mathcal{Z}$*,* $\mathsf{PC.IsValid}$ *is a PPT algorithm that returns a decision bit.*
- $\mathsf{PC.Open}$*: Is a public-coin succinct interactive argument system* $\langle [P_1, \ldots, P_M], V \rangle$ *in the MPC model, where the statement* $(pp, \mathsf{com}, \vec{\zeta}, y)$ *and witness* $(\mathcal{Y} = \{\mathcal{Y}_i\}_{i \in [M]}, \mathcal{Z} = \{\mathcal{Z}_i\}_{i \in [M]})$*, with respect to the relation*

$$R = \left\{ \Big((pp, \mathsf{com}, \vec{\zeta}, y), (\mathcal{Y}, \mathcal{Z}) \Big) : \begin{array}{c} \mathsf{IsValid}(pp, \mathsf{com}, \mathcal{Y}, \mathcal{Z}) = 1, \ \textit{and} \\ \mathsf{ML}(\mathcal{Y}, \vec{\zeta}) = y \end{array} \right\},$$

where each prover P_i *has input* $(pp, \mathsf{com}, \vec{\zeta}, y, \mathcal{Y}_i, \mathcal{Z}_i)$ *and V has input* $(pp, \mathsf{com}, \vec{\zeta}, y)$*.*

In the following sections, we assume that $\mathsf{PC.PartialCom}$ works even if we are given streaming access to $\mathcal{Y}_i$.

We now specify the security properties which are required of PC.

Definition 5 (Multilinear Polynomial Commitment Security). *We require the following three properties from a polynomial commitment scheme:*

– ***Correctness:*** *For every prime p, number of variables n, and all $\mathcal{Y}$ and $\vec{\zeta}$,*

$$\Pr\left[1 = \mathsf{PC.Open}(pp, \mathsf{com}, \vec{\zeta}, y; \mathcal{Y}, \mathcal{Z}) : \begin{array}{c} pp \leftarrow \mathsf{PC.Setup}(1^\lambda, p, 1^n) \\ \{\mathsf{com}_i, \mathcal{Z}_i \leftarrow \mathsf{PC.PartialCom}(pp, \mathcal{Y}_i)\}_{i\in[M]} \\ \mathsf{com}, \mathcal{Z} \leftarrow \mathsf{PC.CombineCom}(pp, \{\mathsf{com}_i\}_{i\in[M]}) \end{array}\right] = 1.$$

– ***Computational Binding:*** *For every prime q, number of variables n, number of parties M, and nonuniform polynomial machine $\mathcal{A}$, there exists a negligible function $\mathsf{negl} : \mathbb{N} \to [0,1]$ such that for every $\lambda \in \mathbb{N}$ and every $z \in \{0,1\}^*$, following holds:*

$$\Pr\left[\begin{array}{c} b_0 = 1 \\ b_1 = 1 \\ \mathcal{Y}_0 \neq \mathcal{Y}_1 \end{array} : \begin{array}{c} pp \leftarrow \mathsf{PC.Setup}(1^\lambda, q, 1^n, M) \\ (\mathsf{com}, \mathcal{Y}_0, \mathcal{Y}_1, \mathcal{Z}_0, \mathcal{Z}_1) \leftarrow \mathcal{A}(1^\lambda, pp, z) \\ b_0 \leftarrow \mathsf{PC.IsValid}(pp, \mathsf{com}, \mathcal{Y}_0, \mathcal{Z}_0) \\ b_1 \leftarrow \mathsf{PC.IsValid}(pp, \mathsf{com}, \mathcal{Y}_1, \mathcal{Z}_1) \end{array}\right] < \mathsf{negl}(\lambda).$$

– ***Properties of*** $\mathsf{PC.Open}$***:*** *The argument* $\mathsf{PC.Open}$ *satisfies the efficiency, completeness and witness-extended emulation properties defined in Sect. 3.*

Looking ahead, in Sect. 6, we will prove the following theorem, showing the existence of a scheme PC which satisfies the properties above.

Theorem 2. *Assume $\mathcal{G}$ is a group sampler where the Hidden Order Assumption holds. Let n be the number of variables, $M \leq 2^n$ be the number of parties. Then, the scheme defined in Sect. 6.2 is a polynomial commitment scheme (as in Sect. 4) for n variate multilinear polynomials over finite field of prime-order p in the MPC model with M parties with the following efficiency guarantees:*

1. $\mathsf{PC.PartialCom}$ *outputs a partial commitment of size* $\mathsf{poly}(\lambda)$ *bits, runs in time* $2^n \cdot \mathsf{poly}(\lambda, n, \log(p))$*, and uses a single pass over the stream.*
2. $\mathsf{PC.PartialEval}$ *outputs a partial evaluation of size* $\lceil\log(p)\rceil$*, runs in time* $(2^n/M) \cdot \mathsf{poly}(n, \log(p))$*, and uses a single pass over the stream.*
3. $\mathsf{PC.CombineCom}$ *and* $\mathsf{PC.CombineEval}$ *have* $O(\log(M))$ *rounds, and each party in it requires* $\mathsf{poly}(\lambda)$ *bits of space.*
4. $\mathsf{PC.Open}$ *takes* $O(n \cdot \log(M))$ *rounds with* $\mathsf{poly}(n, \lambda, \log(p), \log(M))$ *communication.*
5. *The verifier in* $\mathsf{PC.Open}$ *runs in time* $\mathsf{poly}(\lambda, n, \log(p))$*.*
6. *Each party P_i in* $\mathsf{PC.Open}$ *runs in time* $2^n \cdot \mathsf{poly}(n, \lambda, \log(p))$*, requires space* $n \cdot \mathsf{poly}(\lambda, \log(p), \log(M))$*, and uses* $O(n)$ *passes over its stream.*

5 Constructing Succinct Arguments in the MPC Model

Our construction uses the subprotocols $\mathsf{Distribute}$, $\mathsf{Combine}$, and $\mathsf{CalcMerkleTree}$ introduced in [35]. These protocols take $O(\log_\nu M)$ rounds and the communication is $O(S \cdot \nu)$ per round for each machine, for small integral branching factor $\nu \geq 2$.

5.1 Tools from Prior Work

We import two major tools from previous work. The first is the following lemma, which says that any RAM program can be transformed into a circuit C, where the wire assignments of C can be streamed in time and space both proportional to the time and space of the RAM program, respectively. In addition, the circuit logic can be represented succinctly by low-degree polynomials which have properties amenable to sumcheck arguments.

Lemma 1 (From Blumberg et al. [23]). *Let M be an arbitrary (non-deterministic) RAM program that on inputs of length n runs in time $T(n)$ and space $S(n)$. M can be transformed into an equivalent (non-deterministic) arithmetic circuit C over a field $\mathbb{F}$ of size $\mathsf{polylog}(T(n))$. Moreover, there exist cubic extensions $\widehat{\mathsf{add}}$ and $\widehat{\mathsf{mult}}$ of the wiring predicates add and mult of C that satisfy:*

1. *C has size $O(T(n) \cdot \mathsf{polylog}(T(n))$.*
2. *The cubic extensions $\widehat{\mathsf{add}}$ and $\widehat{\mathsf{mult}}$ of C can be evaluated in time $O(\mathsf{polylog}(T(n)))$.*
3. *an (input,witness) pair (x, w) that makes M accept can be mapped to a correct transcript W for C in time $O(T(n) \cdot \mathsf{polylog}(T(n))$ and space $O(S(n)) \cdot \mathsf{polylog}(T(n))$. Furthermore, w is a substring of the transcript W, and any correct transcript W' for C possesses a witness w' for (M, x) as a substring.*
4. *C can be evaluated "gate-by-gate" in time $O(T(n) \cdot \mathsf{polylog}(T(n)))$ and space $O(S(n) \cdot \mathsf{polylog}(T(n)))$.*
5. *The prover's sumcheck messages can be computed in space $O(S(n) \cdot \mathsf{polylog}(T(n)))$.*

5.2 Notation

We make the following notational assumptions about the MPC algorithm Π_L which verifies membership in L.

Let R be the number of rounds that Π_L takes. In each round $r \in [R]$ of an execution of Π_L, the behavior of party $i \in [M]$ is described as a succinct RAM program $\mathsf{NextSt}(i, r, \cdot)$. Thus the program NextSt is a succinct representation of the entire protocol Π_L. We assume NextSt has size much less than S. For convenience, we write $\mathsf{NextSt}_{i,r}(\cdot) = \mathsf{NextSt}(i, r, \cdot)$. We assume that $\mathsf{NextSt}_{i,r}$ takes a string $\mathsf{st}_{i,r-1}||\mathsf{msg}^{\mathsf{in}}_{i,r-1}$ as an input and outputs string $\mathsf{st}_{i,r}||\mathsf{msg}^{\mathsf{out}}_{i,r}$, where $\mathsf{st}_{i,r}$ is the internal, private state of party i in round r and $\mathsf{msg}^{\mathsf{in}}_{i,r-1}$ is the list of messages which party i received in round $r-1$, and $\mathsf{msg}^{\mathsf{out}}_{i,r}$ are the outgoing messages of party i in round r. Note that the space of each party is limited to S bits, so in particular $|\mathsf{st}_{i,r}||\mathsf{msg}^{\mathsf{in}}_{i,r}||\mathsf{msg}^{\mathsf{out}}_{i,r}| \leq S$ for each $i \in [M]$ and $r \in [R]$. We assume that the first-round private state $\mathsf{st}_{i,0}$ of each party i is equal to its private input (x, w_i) (or x if $i = M+1$). In addition, we assume that $\mathsf{msg}^{\mathsf{out}}_{i,r} = \{(j, \ell_j, m_j)\}_j$, where each triple (j, ℓ_j, m_j) means that party i should send message m_j to party j, and that party j should store this message at position ℓ_j in $\mathsf{msg}^{\mathsf{in}}_{j,r-1}$. Finally, we assume that if r is the final round then P_1 writes 1 to the first position of $\mathsf{st}_{1,r}$ iff $x \in L$.

5.3 The Construction

The main construction of a succinct argument in the MPC model works as follows. First, we construct a succinct argument for the following scenario. Fix a round r and corresponding starting states $\mathsf{st}_{i,r-1}||\mathsf{msg}^{\mathsf{in}}_{i,r-1}$ for each party $i \in [M]$, and let π_{r-1} be a Merkle commitment to the concatenation of all these starting states. Let $\mathsf{st}_{i,r}||\mathsf{msg}^{\mathsf{out}}_{i,r}||\mathsf{msg}^{\mathsf{in}}_{i,r}$ be the state of party i after an honest execution of round r, and let π_r be a Merkle commitment to the concatenation of all these end states. Assuming V has x, π_{r-1}, and π_r, the goal is to convince V that π_r is a commitment to states which have been obtained by an honest round-r interaction, starting with the states committed to by π_{r-1}. If we construct an argument for this language, and this argument satisfies witness-extended emulation, this is sufficient for achieving an argument system which verifies an honest execution of the full protocol Π_L with respect to a witness for L.

In the following, we construct such a "round verification protocol," which is our main technical contribution. In Sect. 5.4, we show how to use this round verification protocol to build an argument system for L.

To start, we define a new machine, which we call NextSt'. As before, we write $\mathsf{NextSt}'_{i,r}(\cdot) = \mathsf{NextSt}'(i, r, \cdot)$. Let

$$\mathsf{NextSt}'_{i,r}(\pi_{r-1}, \pi_r, \mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}, \rho_{i,r-1}, \rho_{i,r}, \{\rho_{i\to j,r}\}_j) = 1$$

if the all following holds:

- $\rho_{i,r-1}$ is an opening of π_{r-1} to $(\mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1})$ at position i,
- $\rho_{i,r}$ is an opening of π_r to $\mathsf{st}_{i,r}||\mathsf{msg}^{\mathsf{out}}_{i,r}||\mathsf{msg}^{\mathsf{in}}_{i,r}$ at position i, where

$$\mathsf{NextSt}_{i,r}(\mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}) = \mathsf{st}_{i,r}||\mathsf{msg}^{\mathsf{out}}_{i,r}||\mathsf{msg}^{\mathsf{in}}_{i,r},$$

- Writing $\mathsf{msg}^{\mathsf{out}}_{i,r}$ as $\{(j, \ell_j, m_j)\}_j$, for each j, $\rho_{i\to j,r}$ is an opening to m_j at position ℓ_j in $\mathsf{msg}^{\mathsf{in}}_{j,r}$.

Otherwise, let

$$\mathsf{NextSt}'_{i,r}(\pi_{r-1}, \pi_r, \mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}, \rho_{i,r-1}, \rho_{i,r}, \{\rho_{i\to j,r}\}_j) = 0.$$

Note that since $\mathsf{NextSt}_{i,r}$ is succinct, $\mathsf{NextSt}'_{i,r}$ is also succinct. Let $C_{i,r}$ be the circuit corresponding to $\mathsf{NextSt}'_{i,r}$ via Lemma 1. Also from Lemma 1, party i can stream the gate assignments $W_{i,r}$ of $C_{i,r}$ in space proportional to the space taken by an execution of $\mathsf{NextSt}'_{i,r}$.

We take an approach inspired by that of [22,23] in constructing a sumcheck polynomial that encodes the computation, and using a polynomial commitment to allow for a succinct verifier. Let $s = \lceil \log T' \rceil$, where T' is the number of wires in $C_{i,r}$ (which is constant across i and r). We can index every wire in $C_{i,r}$ with some string $\vec{x} \in \{0,1\}^s$. Define the polynomial $\hat{W}_{i,r}(X_1, \ldots, X_s)$ to be the multilinear extension of $W_{i,r}$, i.e., for all $\vec{x} \in \{0,1\}^s$, $W_{i,r}(\vec{x})$ is the value that $W_{i,r}$ assigns to wire $\vec{x}$. Now, letting $m = \lceil \log M \rceil$, we can index each party by a string $\vec{z} \in \{0,1\}^m$. Define $\hat{W}_r(X_1, \ldots, X_s, Z_1, \ldots, Z_m)$ to be the multilinear polynomial such that $\hat{W}_r(\vec{x}, \vec{z}) = \hat{W}_{i,r}(\vec{x})$, where i is the index which corresponds to $\vec{z}$. Let $\widehat{\mathsf{add}}(X_1, \ldots, X_{3s})$ be the succinct, low-degree polynomial from Lemma 1 where $\widehat{\mathsf{add}}(\vec{x}_1, \vec{x}_2, \vec{x}_3) = 1$ if in $C_{i,r}$ the unique gate which has input wires $\vec{x}_1$ and $\vec{x}_2$ and output wire $\vec{x}_3$ is an addition gate. Note that $\widehat{\mathsf{add}}$ does not depend on i (or r for that matter) since, except for some hardcoded input wires, $C_{i,r} = C_{i',r'}$ for all i, i', r, r'. Similarly, define $\widehat{\mathsf{mult}}(X_1, \ldots, X_{3\,s})$. Finally, define $\widehat{\mathsf{inout}}(X_1, \ldots, X_{3s})$ so that $\widehat{\mathsf{inout}}(\vec{x}_1, \vec{x}_2, \vec{x}_3) = 1$ if either $\vec{x}_3$ is an input wire which is known by V, or $\vec{x}_3$ is an output wire which is known by V and $\vec{x}_1$ and $\vec{x}_2$ are the input wires for the gate whose output wire is $\vec{x}_3$. Define $\hat{I}(X_1, \ldots, X_m)$ to be the multilinear polynomial such that $\hat{I}(\vec{x})$ is the corresponding bit of π_{r-1} (or π_r) if $\vec{x}$ is an input wire which takes the value of a bit of π_{r-1} (or π_r, respectively), and is the corresponding bit of the statement to the argument system if $r = 0$ $\vec{x}$ is an input wire which takes the value of the statement, and is 1 if $\vec{x}$ is an output wire which V knows should be 1.

Given above, we can define the polynomial g as follows:

$$\begin{aligned} g(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = {} & \widehat{\mathsf{add}}(\vec{X}_1, \vec{X}_2, \vec{X}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - (\hat{W}_r(\vec{X}_1, \vec{Z}) + \hat{W}_r(\vec{X}_2, \vec{Z}))) \\ & + \widehat{\mathsf{mult}}(\vec{X}_1, \vec{X}_2, \vec{X}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - (\hat{W}_r(\vec{X}_1, \vec{Z}) \cdot \hat{W}_r(\vec{X}_2, \vec{Z}))) \\ & + \widehat{\mathsf{inout}}(\vec{X}_1, \vec{X}_2, \vec{X}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - \hat{I}(\vec{X}_3)). \end{aligned}$$

With this definition, g vanishes on all boolean inputs if and only if $\hat{W}_r$ encodes transcripts of the correct computations of each party i with respect to starting states committed to in π_{r-1} and ending states committed to in π_r, and if all messages sent by i have been stored in the respective $\mathsf{msg}^{\mathsf{in}}_{j,r}$. For $q \in \mathbb{Z}_p$, let $h_q(\vec{X}) = g(\vec{X}) \cdot \prod_{\beta \in [m+3\,s]}(1 - (1 - q^{2^{\beta-1}})X_i)$. Then, $h_q(\vec{x}) = g(\vec{x}) \cdot q^{\mathsf{bin}^{-1}(\vec{x})}$ for all $\vec{x} \in \{0,1\}^{m+3\,s}$, where $\mathsf{bin}^{-1}(\vec{X})$ is the integer represented by the binary representation $\vec{X}$. We now have defined the polynomials required for the protocol below. If $P_1, \ldots P_M$ can collectively construct the prover's sumcheck messages for the polynomial h_q for a randomly chosen q, then this is sufficient to build an argument that convinces V that g vanishes on the boolean hypercube. We now describe the protocol, assuming the provers have an efficient subprotocol $\mathsf{CalcSumcheckProverMsg}$ (defined below) for constructing their responses. This protocol is heavily inspired by the IOP in [22]. However, that protocol was significantly simpler, since in their setting, there is only one prover who can stream the whole polynomial $\hat{W}_r$. In contrast, we have the task of showing that it is possible to construct the prover's sumcheck responses in a round-efficient way, even given that $\hat{W}_r$ is spread across many different machines.

$\mathsf{VerifyRound}$: **Protocol to verify correctness of one round of Π_L.**

Parameters: Tree fan-in $\nu = s/\log N$.

Inputs: P_i has input $(r, \pi_{r-1}, \pi_r, \mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}, \rho_{i,r-1}, \rho_{i,r}, \{\rho_{i\to j,r}\}_j)$. V has input (x, r, π_{r-1}, π_r). In addition, all parties have the setup α for a polynomial commitment scheme.

Execution:

1. Independently in parallel, each prover computes $\phi_i = \mathsf{PC.PartialCom}(\alpha, \hat{W})$, where for each $\vec{x} \in \{0,1\}^s$, $\hat{W}(\vec{x})$ is the wire assignment for wire $\vec{x}$ in $C_{i,r}$. By Lemma 1, the wire assignments can be computed in a streaming fashion, and $\mathsf{PC.PartialCom}$ works given streaming access to $\hat{W}$.
2. $\mathsf{com} \leftarrow \mathsf{PC.CombineCom}(\alpha, \{\phi_i\}_{i\in[M]})$, so that each party obtains com, the commitment of the polynomial $\hat{W}_r$ defined above.
3. P_1 sends this commitment com to V.
4. V chooses $q \overset{\$}{\leftarrow} \mathbb{F}$ and sends q to P_1.
5. The provers run $\mathsf{Distribute}_\nu(q)$ so that every prover obtains q.
6. The parties now run the sumcheck protocol with respect to h_q defined above. Set $y_1 \leftarrow 0$. For each $\gamma \in [m+3s]$:
   (a) Provers run the subprotocol $\mathsf{CalcSumcheckProverMsg}$ to generate sumcheck prover's message f_γ. P_1 sends f_γ to V.
   (b) V checks that $\deg f_\gamma = \deg_\gamma h_q$, and halts and outputs 0 if the degrees are different.
   (c) V checks that $f_\gamma(0) + f_\gamma(1) = y_\gamma$, and halts and outputs 0 if the equality does not hold.
   (d) V then chooses $\zeta_\gamma \overset{\$}{\leftarrow} \mathbb{F}$, sets $y_{\gamma+1} \leftarrow f_\gamma(\zeta_\gamma)$, and sends ζ_γ to P_1.
7. Write $\vec{\zeta} = (\vec{\zeta_1}, \vec{\zeta_2}, \vec{\zeta_3}, \vec{\zeta_4})$. The provers and verifier run $\mathsf{PC.Open}$ with respect to statements $(\alpha, \mathsf{com}, (\vec{\zeta_1}, \vec{\zeta_4}), \hat{W}_r(\vec{\zeta_1}, \vec{\zeta_4}))$, $(\alpha, \mathsf{com}, (\vec{\zeta_2}, \vec{\zeta_4}), \hat{W}_r(\vec{\zeta_2}, \vec{\zeta_4}))$, and $(\alpha, \mathsf{com}, (\vec{\zeta_3}, \vec{\zeta_4}), \hat{W}_r(\vec{\zeta_3}, \vec{\zeta_4}))$. V halts and outputs 0 if any of these protocols fail to verify.
8. V uses the openings to compute $h_q(\vec{\zeta})$. It checks whether $h_q(\vec{\zeta}) = y_{m+3s}$; if the equality does not hold, V outputs 0; otherwise, it outputs 1.

The $\mathsf{CalcSumcheckProverMsg}$ subprotocol

We now show how the parties $P_1, \ldots, P_M$ can generate the sumcheck prover's polynomials f_γ in a round- and space-efficient manner. For each round γ, the honest $f_\gamma(X)$ is defined to be the following univariate polynomial:

$$f_\gamma(X) = \sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} h_q(\vec{\zeta}, X, \vec{x}),$$

for the random vector $\vec{\zeta}$ chosen by the verifier in previous rounds. (In round one, $\vec{\zeta}$ is the empty vector of length 0.) Recall that $h_q(\vec{X}) = g(\vec{X}) \cdot \prod_{i\in[m+3\,s]}(1 - (1 - q^{2^i} X_i))$, for g as defined below (setting $\vec{X} = (\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z})$):

$$\begin{aligned} g(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = {} & \widehat{\mathsf{add}}(\vec{X}_1, \vec{X}_2, \vec{X}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - (\hat{W}_r(\vec{X}_1, \vec{Z}) + \hat{W}_r(\vec{X}_2, \vec{Z}))) \\ & + \widehat{\mathsf{mult}}(\vec{X}_1, \vec{X}_2, \vec{X}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - (\hat{W}_r(\vec{X}_1, \vec{Z}) \cdot \hat{W}_r(\vec{X}_2, \vec{Z}))) \\ & + \widehat{\mathsf{inout}}(\vec{x}_1, \vec{x}_2, \vec{x}_3)(\hat{W}_r(\vec{X}_3, \vec{Z}) - \hat{I}(\vec{X}_3)). \end{aligned}$$

Observe that $h_q(\vec{X})$ can be written as

$$h_q(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = \sum_{i=j}^{5} p_j(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}),$$

where $p_5(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = \widehat{\mathsf{inout}}(\vec{x}_1, \vec{x}_2, \vec{x}_3) \cdot \hat{I}(\vec{X}_3)$ and can be computed locally by each party,

$$p_4(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = p'_4(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z})\hat{W}_r(\vec{X}_1, \vec{Z})\hat{W}_r(\vec{X}_2, \vec{Z}),$$

and for all $j \in \{1, 2, 3\}$

$$p_j(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = p'_j(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z})\hat{W}_r(\vec{X}_j, \vec{Z}) \ .$$

Here each p'_j is a succinct low-degree polynomial known by V. Thus, to compute the polynomial $f_\gamma(X)$ in small rounds and space, it is sufficient to compute

$$\sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p_j(\vec{\zeta}, X, \vec{x}) \tag{1}$$

in small rounds and space for each $j \in [4]$ (and p_5 locally) and sum the results.

We now show how to do this, focusing first on the case of $i \in \{1, 2, 3\}$. Note that in every round except the first, computing the sum in Eq. (1) involves computing $O(2^{|\vec{x}|})$ interpolations of $\hat{W}_r$. Since the evaluations of $\hat{W}_r$ are distributed among the M parties $P_1, \ldots, P_M$, doing these interpolations requires communication among these parties. If we interpolated $p_j(\vec{\zeta}, X, \vec{x})$ for each $\vec{x}$ and then summed the result, then even if the communication per interpolation is a constant number of rounds, this would mean that computing Eq. (1) would involve a number of rounds linear in the total computation time. So we need something slightly more clever than the naive strategy.

Before we go on, we note that for Eq. (1), it suffices to compute

$$\sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p_j(\vec{\zeta}, \zeta', \vec{x}),$$

for each $\zeta' \in \{0, \ldots, \delta\}$, where δ is the degree of p_j. Once we have these $\delta+1$ field elements, we can interpolate Eq. (1) in constant space. So we focus on computing this; i.e., we focus on computing the following for an arbitrary $\vec{\zeta} \in \mathbb{F}^\gamma$

$$\sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p_j(\vec{\zeta},\vec{x}). \tag{2}$$

Note that each term in the sum above is of the form $p'_j(\vec{\zeta},\vec{x})\hat{W}_r(\vec{\zeta}',\vec{x}')$, where $\vec{\zeta}'$ is obtained from $\vec{\zeta}$ by deleting some (possibly zero) indices, and $\vec{x}'$ is obtained from $\vec{x}$ in the same manner. The key insight which allows us to compute Eq. (2) in low rounds is as follows. Imagine that $\vec{\zeta}' = (\zeta_1)$ is a single element. Then, by the multilinearity of $\hat{W}_r$, it follows that $\hat{W}_r(\zeta_1,\vec{x}') = \zeta_1\cdot\hat{W}_r(1,\vec{x}')+(1-\zeta_1)\cdot\hat{W}_r(0,\vec{x}')$. In the same way, if $\vec{\zeta}' = (\zeta_1,\zeta_2)$, then

$$\begin{aligned}\hat{W}_r(\zeta_1,\zeta_2,\vec{x}') &= \zeta_1\cdot\hat{W}_r(1,\zeta_2,\vec{x}') + (1-\zeta_1)\cdot\hat{W}_r(0,\zeta_2,\vec{x}')\\ &= \zeta_1\cdot\Big(\zeta_2\cdot\hat{W}_r(1,1,\vec{x}') + (1-\zeta_2)\cdot\hat{W}_r(1,0,\vec{x}')\Big)\\ &+ (1-\zeta_1)\cdot\Big(\zeta_2\cdot\hat{W}_r(0,1,\vec{x}') + (1-\zeta_2)\cdot\hat{W}_r(0,0,\vec{x}')\Big)\,.\end{aligned}$$

By a simple use of induction, we can write $\hat{W}_r(\vec{\zeta}',\vec{x}')$, for arbitrary $\vec{\zeta}'$, as

$$\hat{W}_r(\vec{\zeta}',\vec{x}') = \sum_{\vec{y}\in\{0,1\}^{|\vec{\zeta}'|}} c_{\vec{\zeta}',\vec{y}}\cdot\hat{W}_r(\vec{y},\vec{x}') \tag{3}$$

where

$$c_{\vec{\zeta}',\vec{y}} = \prod_{j=1}^{|\vec{\zeta}'|}(\zeta_j\cdot y_j + (1-\zeta_j)(1-y_j)) = \prod_{j=1}^{|\vec{\zeta}'|}\{\zeta_j \text{ if } y_j = 1, \text{ otherwise } (1-\zeta_j)\}\,.$$

It follows that we can rewrite Eq. (2) as

$$\sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p_j(\vec{\zeta},\vec{x}) = \sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p'_j(\vec{\zeta},\vec{x})\left(\sum_{\vec{y}\in\{0,1\}^{|\vec{\zeta}'|}} c_{\vec{\zeta}',\vec{y}}\cdot\hat{W}_r(\vec{y},\vec{x}')\right) \tag{4}$$

$$= \sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}}\ \sum_{\vec{y}\in\{0,1\}^{|\vec{\zeta}'|}} c'_{\vec{x},\vec{\zeta}',\vec{y}}\cdot\hat{W}_r(\vec{y},\vec{x}'), \tag{5}$$

where $c'_{\vec{x},\vec{\zeta}',\vec{y}}$ is computable in space proportional to the space required to compute $c_{\vec{\zeta}',\vec{y}}$ and $p'_j(\vec{\zeta},\vec{x})$.

Since Eq. (2) can be written as a weighted sum of evaluations of $\hat{W}_r$ on points in the boolean hypercube, and since all such evaluations are partitioned across the provers, each prover can compute a component of the sum by streaming the computation in space $O(S)$, and then the provers can all sum their components together using a large-arity tree in constant rounds.

The case where $j = 4$ is more involved. Recall that the goal is to compute

$$\sum_{\vec{x}\in\{0,1\}^{m+3\,s-\gamma}} p_4(\vec{\zeta},\vec{x}), \tag{6}$$

for some given $\vec{\zeta} \in \mathbb{F}^{\gamma}$. We first handle the case where $\gamma \leq 3s$. In this case,

$$\sum_{\vec{x} \in \{0,1\}^{m+3\,s-\gamma}} p_4(\vec{\zeta}, \vec{x}) = \sum_{\vec{z} \in \{0,1\}^m} \sum_{\vec{x}' \in \{0,1\}^{3\,s-\gamma}} p_4'(\vec{\zeta}, \vec{x}', \vec{z}) \hat{W}_r(\vec{\zeta}_1, \vec{z}) \hat{W}_r(\vec{\zeta}_2, \vec{z}), \quad (7)$$

where $\vec{\zeta}_1$ and $\vec{\zeta}_2$ are both a combination of $\vec{\zeta}$ and $\vec{x}$. Observe that from the discussion above, for each $\vec{z} \in \{0,1\}^m$, the values $\{\hat{W}_r(\vec{\zeta}_j)\}_{\vec{\zeta}_j}$ can be streamed by a single party P_i, where $\vec{z}$ is the binary representation of i, by streaming the values of $\hat{W}_r$ in the boolean hypercube and then using Eq. (3). Thus, for each $\vec{z}$, the inner sum $\sum_{\vec{x}' \in \{0,1\}^{3\,s-\gamma}} p_4'(\vec{\zeta}, \vec{x}', \vec{z}) \hat{W}_r(\vec{\zeta}_1, \vec{z}) \hat{W}_r(\vec{\zeta}_2, \vec{z})$ can be computed by a single party in $O(S)$ space. The parties can then sum these terms in a large-arity tree, thus computing Eq. (6) in $O(1)$ rounds and $O(S)$ space.

We now consider the case where $\gamma > 3s$. Write $\gamma = 3\,s + m'$, for some $m' > 1$, and write $\vec{\zeta} = (\vec{\zeta}_1, \vec{\zeta}_2, \vec{\zeta}_3, \vec{\zeta}_4)$. In this case,

$$\sum_{\vec{x} \in \{0,1\}^{m+3\,s-\gamma}} p_4(\vec{\zeta}, \vec{x}) = \sum_{\vec{z}' \in \{0,1\}^{m-m'}} p_4'(\vec{\zeta}, \vec{z}') \hat{W}_r(\vec{\zeta}_1, \vec{\zeta}_4, \vec{z}') \hat{W}_r(\vec{\zeta}_2, \vec{\zeta}_4, \vec{z}'),$$

and then again by Eq. (3), this is equal to

$$\sum_{\vec{z}' \in \{0,1\}^{m-m'}} \mathsf{term}_{\vec{z}'} \quad (8)$$

where $\mathsf{term}_{\vec{z}'}$ is the following:

$$p_4'(\vec{\zeta}, \vec{z}') \left(\sum_{\vec{y}_4^{(1)} \in \{0,1\}^{m'}} c_{\vec{\zeta}_4, \vec{y}_4^{(1)}} \cdot \hat{W}_r(\vec{\zeta}_1, \vec{y}_4^{(1)}, \vec{z}') \right) \left(\sum_{\vec{y}_4^{(2)} \in \{0,1\}^{m'}} c_{\vec{\zeta}_4, \vec{y}_4^{(2)}} \cdot \hat{W}_r(\vec{\zeta}_2, \vec{y}_4^{(2)}, \vec{z}') \right). \quad (9)$$

Note that for any $\hat{W}_r(\vec{\zeta}_j, \vec{y}_4^{(2)}, \vec{z}')$, there is a party (indexed by $(\vec{y}_4^{(2)}, \vec{z}')$ who can compute this value locally, so WLOG, we assume each party has precomputed this corresponding value. Observe that $\vec{z}'$ defines a *subset* of parties, indexed by the set $S_{\vec{z}'} = \{\vec{y}_4, \vec{z}' \colon y_4 \in \{0,1\}^{m'}\}$, and distinct from $S_{\vec{z}''}$ for all $\vec{z}'' \neq \vec{z}'$. Observe also that for each $\vec{z}'$, to compute $\mathsf{term}_{\vec{z}'}$, only the parties in $S_{\vec{z}'}$ must interact, and they can compute the sum in Eq. (9) in constant rounds and $O(S)$ space by first computing the two inner sums via large-arity trees as in all the previous cases, and then multiplying these two summed values together and weighting them according to $p_4'(\vec{\zeta}, \vec{z}')$. Thus, to compute the outer sum, for each $\vec{z}'$, the parties in $S_{\vec{z}'}$ can interact in the manner described above, simultaneously with all other $S_{\vec{z}''}$. Then, once each set has their term of the sum, representative parties for each of the sets can again use a large-arity tree to obtain the final result in constant rounds and $O(S)$ space.

We now give the description of $\mathsf{CalcSumcheckProverMsg}$.

The protocol $\mathsf{CalcSumcheckProverMsg}$**.**

Parameters: Tree fan-in $\nu = s/\log N$.

Inputs: P_i has input $(r, \pi_{r-1}, \pi_r, \mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}, \rho_{i,r-1}, \rho_{i,r}, \{\rho_{i\to j,r}\}_j)$. In addition, all parties have the setup α for a polynomial commitment scheme, a field element $q \in \mathbb{F}$, the round γ of the sumcheck, and the verifier queries $\vec{\zeta}$, where $|\vec{\zeta}| = \gamma - 1$.

Execution:

1. Write $h_q(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = \sum_{i=j}^{5} p_j(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z})$. Party P_1 locally computes $\mathsf{summand}_5 = \sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_5(\vec{\zeta}, X, \vec{x})$, where
$$p_5(\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{Z}) = \widehat{\mathsf{inout}}(\vec{x}_1, \vec{x}_2, \vec{x}_3) \cdot \hat{I}(\vec{X}_3),$$
and stores the result.
2. For $j \in \{1, \dots, 3\}$, the parties compute $\sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_j(\vec{\zeta}, X, \vec{x})$ as:
   (a) For $\zeta' \in \{0, \dots, \deg(p_j)\}$, compute $\sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_j(\vec{\zeta}, \zeta', \vec{x})$ as:
      i. Each party P_i streams the computation of $C_{i,r}$ in order to compute the component $\mathsf{component}_i$ of the sum in eq. (5) which it has access to.
      ii. The parties run the protocol $\mathsf{Combine}_\nu(+, \{\mathsf{component}_i\}_{i\in[M]})$ so that P_1 learns $\sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_j(\vec{\zeta}, \zeta', \vec{x})$.
   (b) Once P_1 has these $\deg(p_j) + 1$ values, it interpolates $\mathsf{summand}_j = \sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_j(\vec{\zeta}, X, \vec{x})$.
3. The parties now compute $\mathsf{summand}_4 = \sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_4(\vec{\zeta}, X, \vec{x})$. If $\gamma \le 3s$, then for each $\zeta' \in \{0, \dots, \deg(p_4)\}$:
   (a) Each party P_i computes the inner sum
$$\mathsf{component}_i = \sum_{\vec{x}'\in\{0,1\}^{3s-\gamma}} p_4'(\vec{\zeta}, \vec{x}', \vec{z})\hat{W}_r(\vec{\zeta}_1, \vec{z})\hat{W}_r(\vec{\zeta}_2, \vec{z})$$
   from eq. (7), where $\vec{z}$ is the index of i in binary form.
   (b) The parties run the protocol $\mathsf{Combine}_\nu(+, \{\mathsf{component}_i\}_{i\in[M]})$ so that P_1 learns $\sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_4(\vec{\zeta}, \zeta', \vec{x})$.
4. On the other hand, if $\gamma > 3s$ then for each $\zeta' \in \{0, \dots, \deg(p_4)\}$:
   (a) For each $\vec{z}' \in \{0,1\}^{m-m'}$, the parties in the set $S_{\vec{z}'}$ do the following to compute $\mathsf{term}_{\vec{z}'}$:
      i. Each party P_i in $S_{\vec{z}'}$ computes the component $\mathsf{component}_i$ of $\left(\sum_{\vec{y}_4^{(1)}\in\{0,1\}^{m'}} c_{\vec{\zeta}_4, \vec{y}_4^{(1)}} \cdot \hat{W}_r(\vec{\zeta}_1, \vec{y}_4^{(1)}, \vec{z}')\right)$ which it has access to.
      ii. The parties in $S_{\vec{z}'}$ run $\mathsf{Combine}_\nu(+, \{\mathsf{component}_i\}_{i\in S_{\vec{z}'}})$ so that the lexicographically first party in $S_{\vec{z}'}$ learns factor_1.

iii. Each party P_i in $S_{\vec{z}'}$ computes the component $\mathsf{component}_i$ of $\left(\sum_{\vec{y}_4^{(2)}\in\{0,1\}^{m'}} c_{\zeta_4,\vec{y}_4^{(2)}} \cdot \hat{W}_r(\vec{\zeta}_2, \vec{y}_4^{(2)}, \vec{z}')\right)$ which it has access to.

iv. The parties in $S_{\vec{z}'}$ run $\mathsf{Combine}_\nu(+, \{\mathsf{component}_i\}_{i\in S_{\vec{z}'}})$ so that the lexicographically first party in $S_{\vec{z}'}$ learns factor_2.

v. The lexicographically first party in $S_{\vec{z}'}$ now computes $\mathsf{term}_{\vec{z}'} = p_4'(\vec{\zeta}, \vec{z}') \cdot \mathsf{factor}_1 \cdot \mathsf{factor}_2$ locally, by eq. (9).

(b) Let $P_{i_{\vec{z}'}}$ be the lexicographically first party in $S_{\vec{z}'}$. The parties $\{P_{i_{\vec{z}'}} : \vec{z}' \in \{0,1\}^{m-m'}\}$ runs $\mathsf{Combine}_\nu(+, \{\mathsf{term}_{\vec{z}'}\}_{\vec{z}'\in\{0,1\}^{m-m'}})$ so that P_1 learns $\sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_4(\vec{\zeta}, \zeta', \vec{x})$, as in eq. (8).

5. Once P_1 has these $\deg(p_j) + 1$ values, it interpolates them to compute $\mathsf{summand}_4 = \sum_{\vec{x}\in\{0,1\}^{m+3s-\gamma}} p_4(\vec{\zeta}, X, \vec{x})$.
6. P_1 outputs $\sum_{j=1}^{5} \mathsf{summand}_j$.

Efficiency

We now discuss the efficiency of the $\mathsf{VerifyRound}$ protocol.

Round complexity. The protocol $\mathsf{VerifyRound}$ can be separated into two steps: first, the provers commit to the polynomial $\hat{W}_r$ and receive a random q from V, and then second, the parties carry out a sumcheck protocol. The first step is dominated by the subprotocols $\mathsf{PC.CombineCom}(\alpha, \{\phi_i\}_{i\in[M]})$ and $\mathsf{Distribute}_\nu(q)$. Note that since $\nu = \lambda$, and each of these two protocols take $O(\log_\nu(M))$ rounds, the first step takes a constant number of rounds. The second step takes $(m + 3\,s) \cdot (R_{\mathsf{CalcSumcheckProverMsg}} + C_1) + C_2 \cdot R_{\mathsf{PC.Open}}$ rounds, where $m + 3s$ is $\mathsf{polylog}(N)$, $R_{\mathsf{CalcSumcheckProverMsg}}$ and $R_{\mathsf{PC.Open}}$ are the number of rounds required for the $\mathsf{CalcSumcheckProverMsg}$ and $\mathsf{PC.Open}$ subprotocols respectively, and C_1 and C_2 are constants. As explained in Sect. 6, $R_{\mathsf{PC.Open}} = \mathsf{polylog}(|\hat{W}_r|)$, which is $\mathsf{polylog}(N)$. As explained in Sect. 5.3, $R_{\mathsf{CalcSumcheckProverMsg}}$ is constant. Thus, $(m + 3\,s) \cdot (R_{\mathsf{CalcSumcheckProverMsg}} + C_1) + C_2 \cdot R_{\mathsf{PC.Open}}$ is $\mathsf{polylog}(N)$. It follows that the entire protocol $\mathsf{VerifyRound}$ takes $\mathsf{polylog}(N)$ rounds.

Space complexity per party. By the properties of the polynomial commitment and the sumcheck protocol, the verifier takes space $\mathsf{polylog}(N) \cdot \mathsf{poly}(\lambda)$. The provers each take space $S \cdot \mathsf{poly}(\lambda)$; this follows from the following:

– Each party's polynomial $\hat{W}$ which encodes the wire assignments of $C_{i,r}$ can be streamed in space $O(S)$ by Lemma 1, and $\mathsf{PC.PartialCom}$ works assuming streaming access to $\hat{W}$.
– $\mathsf{PC.CombineCom}$ and $\mathsf{PC.Open}$ are MPC protocols where the provers require at most $S \cdot \mathsf{poly}(\lambda)$ space, as per the properties of PC.
– $\mathsf{CalcSumcheckProverMsg}$ is an MPC protocol where the provers require at most $S \cdot \mathsf{poly}(\lambda)$ space, as discussed in the previous section.

5.4 From Round Verification to a Full Argument

In this section, we use the $\mathsf{VerifyRound}$ protocol from Sect. 5 and the polynomial commitment PC from Sect. 6.2 to achieve a succinct argument for a language L, assuming L has a MPC verification algorithm Π_L as described in Sect. 5.2.

The formal description of the argument system is as follows. Assume the original protocol Π_L runs for R rounds.

The argument system for L.

Parameters: Tree fan-in $\nu = s/\log N$.

Inputs: P_i has input (x, w_i). V has input x. In addition, all parties have the CRS α for a polynomial commitment scheme.

Execution:

1. V samples a hash key h and sends it to P_1. The provers run $\mathsf{Distribute}_\nu(h)$.
2. Each P_i sets $\mathsf{st}_{i,0} = (x, w_i)$, and sets $\mathsf{msg}^{\mathsf{in}}_{i,0}$ to be the empty string.
3. The provers run the subprotocol $\mathsf{CalcMerkleTree}_h(\{\mathsf{st}_{i,0}\}_{i\in[M]})$ so that each P_i learns a Merkle root π_0 along with an opening $\rho_{i,0}$ for $\mathsf{st}_{i,r}$.
4. P_1 sends π_0 to V.
5. For each round $r \in [R]$, the parties do the following:
   (a) Each P_i runs $\mathsf{NextSt}_{i,r}(\mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1})$ to obtain $\mathsf{st}_{i,r}$ and $\mathsf{msg}^{\mathsf{out}}_{i,r}$.
   (b) For prover P_i, for each triple $(j, \ell_j, m_j) \in \mathsf{msg}^{\mathsf{out}}_{i,r}$, P_i sends (ℓ_j, m_j) to prover P_j, who stores m_j at position ℓ_j in $\mathsf{msg}^{\mathsf{in}}_{j,r}$.
   (c) The provers run the subprotocol $\mathsf{CalcMerkleTree}_h$ on inputs $(\{(\mathsf{st}_{i,r}, \mathsf{msg}^{\mathsf{out}}_{i,r}, \mathsf{msg}^{\mathsf{in}}_{i,r})\}_{i\in[M]})$ so that each P_i learns a Merkle root π_r along with an opening $\rho_{i,r}$ for $(\mathsf{st}_{i,r}, \mathsf{msg}^{\mathsf{out}}_{i,r}, \mathsf{msg}^{\mathsf{in}}_{i,r})$.
   (d) P_1 sends π_r to V.
   (e) For prover P_i, for each message in $\mathsf{msg}^{\mathsf{in}}_{i,r}$, P_i sends an opening $\rho_{i\to j,r}$ of that position in $\mathsf{msg}^{\mathsf{in}}_{i,r}$ to the sender P_j of that message.
   (f) The parties run the subprotocol $\mathsf{VerifyRound}$, where each prover P_i has input $(r, \pi_{r-1}, \pi_r, \mathsf{st}_{i,r-1}, \mathsf{msg}^{\mathsf{in}}_{i,r-1}, \rho_{i,r-1}, \rho_{i,r}, \{\rho_{i\to j,r}\}_j)$, and the verifier V has input (x, r, π_{r-1}, π_r).
   (g) If $\mathsf{VerifyRound}$ aborts, then V aborts and rejects.
6. P_1 sends an opening ρ of the first position of $\mathsf{st}_{i,R}$ w.r.t. π_R to V.
7. V accepts if the opening bit is 1, and rejects otherwise.

Efficiency. The round complexity of the above argument is $R\cdot\mathsf{polylog}(N)$, where R is the number of rounds taken by Π_L. The space complexity is $S \cdot \mathsf{poly}(\lambda)$ per party. The round and space complexity of the argument follows from those of $\mathsf{VerifyRound}$ discussed above.

Security. We have the following theorem and defer its proof to the full version.

Theorem 3. *Assume the polynomial commitment scheme* PC *satisfies the security properties in Definition 5. Then the argument system above satisfies witness-extended emulation with respect to the language L.*

6 Constructing Polynomial Commitments in the MPC Model

Our construction extensively uses the polynomial commitment scheme of Block et al. [22], which we describe in detail in the full version. To describe our construction, we first introduce the *distributed streaming model* in Sect. 6.1, then describe the construction in Sect. 6.2 with its proof in Sect. 6.3.

6.1 Distributed Streaming Model

Looking ahead to our goal of designing succinct arguments in the MPC model, we consider an enhancement of the streaming model [22] to the MPC setting. We refer to the model as the *distributed streaming model*: Let $\mathcal{Y} \in \mathbb{F}^N$ be some multilinear polynomial and let $\{\mathcal{Y}_i \in \mathbb{F}^{N/M}\}_{i\in[M]}$ be the set of partial descriptions vectors such that $\mathcal{Y} = \mathcal{Y}_1 \mid\mid \mathcal{Y}_2 \mid\mid \ldots \mid\mid \mathcal{Y}_M$. In the distributed streaming model, we assume that each of the S-space bounded parties P_i have streaming access only to the elements of their partial description vector $\mathcal{Y}_i$, where $S \ll N/M$.

While adapting Block et al. [22] to the distributed streaming model, we need to ensure two properties: (a) *low-space provers* and (b) a *low-round protocol.* A naive low space implementation is achieved by blowing up the number of rounds of interaction. Similarly, a naive polylogarithmic round protocol is achieved by simply having each party communicate their whole input (in a single round) to a single party, but this incurs high space for the prover. Achieving the two properties together is the main technical challenge. We build a low-space and a low-round protocol by heavily exploiting the algebraic structure of [22].

6.2 Our New Construction

To support n variate polynomials, recall that each party P_i holds a partial vector $\mathcal{Y}_i$ over $\mathbb{F}$ of size N/M and the corresponding index set $I_i = \{(i-1)\cdot N/M, \ldots, iN/M - 1\}$. The $\mathsf{PC.Setup}$ algorithm is identical to [22], and the $\mathsf{PC.PartialCom}$ and $\mathsf{PC.CombineCom}$ collectively implement the commitment algorithm of [22], and $\mathsf{PC.Open}$ implements their open algorithm.

$\underline{\mathsf{PC.Setup}(1^\lambda, p, 1^n, M):}$ The public parameters pp output by $\mathsf{PC.Setup}$ contains the tuple $(g, p, \mathbb{G})$ where g is a random element of the hidden order group $\mathbb{G}$ and q is a sufficiently large integer odd integer (i.e., $q > p \cdot 2^{n\cdot \mathsf{poly}(\lambda)}$).

$\underline{\mathsf{PC.PartialCom}(pp, \mathcal{Y}_i):}$ Each of the parties locally run this algorithm to compute their partial commitment to the polynomial. In particular, on inputs $pp = (q, g, \mathbb{G})$ and the partial sequence $\mathcal{Y}_i \in \mathbb{F}^{N/M}$, the algorithm $\mathsf{PartialCom}$ outputs a commitment com_i to $\mathcal{Y}_i$ by encoding its elements as an integer in base q. Specifically, $\mathsf{com}_i = g^{z_i}$ where

$$z_i = q^{(i-1)N/M} \left(\sum_{\vec{b} \in \{0,1\}^{n-m}} q^{\vec{b}} \cdot \mathcal{Y}_{i\vec{b}} \right) , \tag{10}$$

and private partial decommitment is the sequence $\mathcal{Z}_i = \mathsf{lift}(\mathcal{Y}_i)$. We give the formal description of this algorithm in the streaming model below.

Protocol 1 $\mathsf{PC.PartialCom}_\nu(pp, \mathcal{Y}_i)$

```
Require: Party $P$ holds a string $pp = (q, g, \mathbb{G})$ where $|pp| \le S$ and has
    streaming access to the elements in the sequence $\mathcal{Y}_i$ in lexicographic order.
Ensure: $P$ party holds $\mathsf{com}$ where $\mathsf{com} = g^{z_i}$ is as defined in Equation (10).
1: Let $\mathsf{com} = 1 \in \mathbb{G}$, $\mathsf{temp} = g$.
2: for $\vec{b} \in \{0,1\}^{n-m}$ do
3:     $\mathsf{com} = \mathsf{com} \cdot \mathsf{temp}^{(\mathcal{Y}_i)_{\vec{b}}}$
4:     $\mathsf{temp} = \mathsf{temp}^q$
5: $\mathsf{com} = \mathsf{com}^{q^{(i-1)N/M}}$.
6: output $\mathsf{com}$
```

<u>$\mathsf{PC.CombineCom}(pp, \{\mathsf{com}_i\})$:</u> Parties each holding their partial commitments com_i want to jointly compute a full commitment $\mathsf{com} = \prod_{i \in [M]} \mathsf{com}_i$. For this, parties run the $\mathsf{Combine}$ subprotocol on their inputs with op as the group multiplication and P_1 as the receiver. Then, P_1 outputs com as the commitment.

<u>$\mathsf{PC.PartialEval}(pp, \mathcal{Y}_i, \vec{\zeta})$:</u> Each of the parties locally run this algorithm to compute their contributions to the evaluation. In particular, on input the CRS pp, a partial vector $\mathcal{Y}_i \in \mathbb{F}^{N/M}$ and a evaluation point $\vec{\zeta} \in \mathbb{F}^n$, the partial evaluation algorithm outputs $y_i \in \mathbb{F}$ such that

$$y_i = \sum_{\vec{b} \in \{0,1\}^{n-m}} \mathcal{Y}_{i\vec{b}} \cdot \overline{\chi}(\vec{\zeta}, \vec{b} + (i-1) \cdot M) . \tag{11}$$

We give the formal description of this algorithm in the streaming model below.

Protocol 2 $\mathsf{PC.PartialEval}_\nu(pp, \mathcal{Y}_i, \vec{\zeta})$

```
Require: Party $P$ holds a string $pp = (q, g, \mathbb{G})$ and $\vec{\zeta}$ where $|pp|, |\vec{\zeta}| \le S$, and
    has streaming access to the elements in $\mathcal{Y}_i$ in lexicographic order.
Ensure: $P$ party holds $y_i$ as defined in Equation (11).
1: Let $y_i = 0 \in \mathbb{F}$.
2: for $\vec{b} \in \{0,1\}^{n-m}$ do
3:     $y_i = y_i + (\mathcal{Y}_i)_{\vec{b}} \cdot \overline{\chi}(\vec{\zeta}, \vec{b} + (i-1) \cdot M)$
4: output $y_i$
```

<u>$\mathsf{PC.CombineEval}(pp, y_i, \vec{\zeta})$:</u> Parties each holding their partial evaluations y_i want to jointly compute the full evaluation $y = \sum_{i \in [M]} y_i$. For this, parties run the $\mathsf{Combine}$ subprotocol on their inputs with the field addition as the associate operator op and P_1 as the receiver. Then, P_1 outputs y as the evaluation.

PC.Open The PC.Open algorithm is the natural adaptation of the Open algorithm in [22] to the distributed streaming model. Specifically, all parties (including V) hold the public parameters $pp = (q, p, \mathbb{G})$, the claimed evaluation $y \in \mathbb{F}$, the evaluation point $\vec{\zeta} \in \mathbb{F}^n$ and the commitment com. Further, each party P_i has streaming access to the entries in its partial decommitment vector $\mathcal{Z}_i$.

The protocol PC.Open**.**

Inputs: Each party P_i holds a string $pp = (q, g, \mathbb{G})$, $\vec{\zeta}$, y and com where $|pp|, |\vec{\zeta}|, |\mathsf{com}|, |y| \leq S$, and has streaming access to the elements in the sequence $\mathcal{Y}_i$ in lexicographic order. The verifier V holds $pp, \vec{\zeta}, \mathsf{com}, y$.

Execution:

1. All parties and the verifier compute the λ-fold repetitions $\vec{\mathsf{com}}^{(0)}$ and $\vec{y}^{(0)}$ of com and y respectively as done in the Open algorithm of [22]. P_i views $\mathcal{Y}_i$ as a vector $\mathcal{Z}_i = \mathsf{lift}(\mathcal{Y}_i)$ over the integers. Further, let $\mathcal{Z} = \mathcal{Z}_1 || \mathcal{Z}_2 || \ldots \mathcal{Z}_M$, and let $Z^{(0)}$ be the λ-fold repetition of $\mathcal{Z}$ as done in the Open algorithm of [22]. By $Z_i^{(0)}$ we denote the part of $Z^{(0)}$ corresponding to $\mathcal{Z}_i$.
2. For $k \in [0, \ldots, n-1]$, do the following:
   (a) Each party P_i, having streaming access to columns in $Z_i^{(0)}$, computes their contribution to $\vec{y}_{\mathsf{L}}^{(k)}$, $\vec{y}_{\mathsf{R}}^{(k)}$, $\vec{\mathsf{com}}_{\mathsf{L}}^{(k)}$ and $\vec{\mathsf{com}}_{\mathsf{R}}^{(k)}$.
   (b) Then, each party run the Combine protocol on their respective contributions such that P_1 learns $\vec{y}_{\mathsf{L}}^{(k)}$, $\vec{y}_{\mathsf{R}}^{(k)}$, $\vec{\mathsf{com}}_{\mathsf{L}}^{(k)}$ and $\vec{\mathsf{com}}_{\mathsf{R}}^{(k)}$. P_1 then forwards these values to the verifier V.
   (c) V checks that $\vec{y}^{(k)} = \vec{y}_{\mathsf{L}}^{(k)} \cdot (1 - \zeta_{k+1}) + \vec{y}_{\mathsf{R}}^{(k)} \cdot \zeta_{k+1}$.
   (d) P_1 and V run a PoE protocol on inputs $(\vec{\mathsf{com}}_{\mathsf{R}}^{(k)}, \vec{\mathsf{com}}^{(k)}/\vec{\mathsf{com}}_{\mathsf{L}}^{(k)}, q, n-k-1, \lambda)$ as in line 9 of MultiEval procedure of [22].
   (e) V samples $U^{(k)} = [U_{\mathsf{L}}^{(k)} \mid\mid U_{\mathsf{R}}^{(k)}] \leftarrow \{0,1\}^{\lambda \times 2\lambda}$ and sends $U^{(k)}$ to P_1 where $U_{\mathsf{L}}^{(k)}, U_{\mathsf{R}}^{(k)} \in \{0,1\}^{\lambda \times \lambda}$.
   (f) P_1 runs the Distribute subprotocol with input $U^{(k)}$ with other P_i's.
   (g) All parties P_i and V locally compute the following:
$$\vec{y}^{(k+1)} = U_{\mathsf{L}}^{(k)} \cdot \vec{y}_{\mathsf{L}}^{(k)} + U_{\mathsf{R}}^{(k)} \cdot \vec{y}_{\mathsf{R}}^{(k)}$$
$$\vec{\mathsf{com}}^{(k+1)} = (U_{\mathsf{L}}^{(k)} \star \vec{\mathsf{com}}_{\mathsf{L}}^{(k)}) \odot (U_{\mathsf{R}}^{(k)} \star \vec{\mathsf{com}}_{\mathsf{R}}^{(k)}) \ .$$
3. Each party P_i computes $Z_i^{(n)}$ where $Z_i^{(n)}$ is obtained by restricting the summation in the expression for $Z^{(n)}$ to I_i.
4. Parties run the Combine protocol on $Z_i^{(n)}$ with the integer addition operation to compute $Z^{(n)}$, and forward to V.
5. V accepts iff $||Z^{(n)}||_\infty \leq p(2\lambda)^n$, $\vec{y}^{(n)} = Z^{(n)} \mod p$, and $\vec{\mathsf{com}}^{(n)} = g^{Z^{(n)}}$.

6.3 Proof of Theroem 2

We now prove Theorem 2 – our main theorem statement for multilinear polynomial commitments in the MPC model. The correctness, binding and witness-extended emulation properties follow readily from that of [22]: this is because, for these properties, it suffices to view the cluster of provers as monolithic. In such a setting, the above described polynomial commitment scheme is then identical to that of [22]. Finally, we argue about the efficiency of each of the algorithms next.

Efficiency of $\mathsf{PC.PartialCom}$**.** In $\mathsf{PC.PartialCom}$ (Sect. 6.2), each party P_i runs through the stream of $\mathcal{Y}_i$ once, and for each of the $2^n/M$ elements performs the following computation: In line 3, it does a single group exponentiation where the exponent is an $\mathbb{F}$ value, and performs a single group multiplicaton. In line 4, it performs a group exponentiation where the exponent is q. Thus, lines 3–4 results in total runtime of $(2^n/M) \cdot \mathsf{poly}(\lambda, \log(p), \log(q))$. On line 5, it performs a single group exponentiation where the exponent is $q^{(i-1)N/M}$ followed by a single group multiplication. The former requires $(i-1)(N/M)\mathsf{poly}(\lambda, \log(q))$ time whereas latter requires $\mathsf{poly}(\lambda)$ run time. Plugging the value of q, results in an overall time of $2^n \cdot \mathsf{poly}(\lambda, n, \log(p))$. The output is a single group elements which require $\mathsf{poly}(\lambda)$ bits, and only one pass over the stream $\mathcal{Y}_i$ is required.

Efficiency of $\mathsf{PC.CombineCom}$**.** Recall from Sect. 6.2, that in $\mathsf{PC.CombineCom}$, all parties run the $\mathsf{Combine}$ subprotocol on local inputs of $\mathsf{poly}(\lambda)$ bits. This requires $O(\log M)$ rounds and each party only requires $\mathsf{poly}(\lambda)$ bits of space.

Efficiency of $\mathsf{PC.PartialEval}$**.** Recall from Sect. 6.2, each party P_i runs through the stream of $\mathcal{Y}_i$ once, and for each of the $2^n/M$ elements performs the following operations in line 3: (a) computes the polynomial $\overline{\chi}$ on inputs of size n, and (b) performs a single field multiplication and addition. Thus $\mathsf{PC.PartialEval}$'s running time is bounded by $(2^n/M) \cdot \mathsf{poly}(\lambda, n, \log(p))$, the output is a single field element of $\lceil \log(p) \rceil$ bits, and only one pass over $\mathcal{Y}_i$ is required.

Efficiency of $\mathsf{PC.CombineEval}$**.** Recall from Sect. 6.2, that in $\mathsf{PC.CombineEval}$, all parties run the $\mathsf{Combine}$ subprotocol on local inputs of $\mathsf{poly}(\lambda)$ bits. This requires $O(\log M)$ rounds and each party only requires $\mathsf{poly}(\lambda)$ bits of space.

Communication/Round Complexity of $\mathsf{PC.Open}$**.** The round complexity of $\mathsf{PC.Open}$ as described in Sect. 6.2 is dominated by line 2. In particular, line 2 is executed for $O(n)$ times where in each iteration k: parties perform local computations in all lines except 2-(b), 2-(d) and 2-(f). In particular, in 2-(b) (resp., 2-(f)), an instantiation of the $\mathsf{Combine}$ (resp., $\mathsf{Distribute}$)subprotocol is run which requires $O(\log(M))$ rounds. Additionally, in 2-(d), party P_1 and the verifier engage in a POE protocol which requires $O(n-k)$ rounds. Therefore, overall, the round complexity of $\mathsf{PC.Open}$ is $O(n \cdot \log(M))$ rounds. In terms of communication complexity, in each round of the protocol at most $\mathsf{poly}(\lambda, n, \log(p), \log(M))$ bits are transmitted, therefore overall its bounded by $\mathsf{poly}(\lambda, n, \log(p), \log(M))$.

The Efficiency of $\mathsf{PC.Open}$**.** The verifier efficiency is dominated by its computation in the PoE execution in line 2 of each of the n rounds, which is bounded by $\mathsf{poly}(\lambda, n, \log(p), \log(q))$. Now onto the prover efficiency. The efficiency of each party P_i is dominated by the n iterative executions of line 2 of the $\mathsf{PC.Open}$(Sect. 6.2). In each iteration: in line 2-(a), P_i runs through the stream of $\mathcal{Y}_i$ once, and for each of the $2^n/M$ elements performs some $\mathsf{poly}(\lambda, n)$ computation for computing the matrices $M_{\vec{c}}$ as well as an $O(n)$ size-product of evaluations of the χ function. Further, the prover computation in lines 2-(d) through 2-(g), doesn't depend on the stream. In particular, its running time is dominated by its computation in lines 2-(d) where P_1 acts as a prover in the PoE protocol where the exponent is of the form $q^{2^{n-k-1}}$. This results in overall running time of $2^n \cdot \mathsf{poly}(\lambda, n, \log(p))$. Further, the prover's space in each of the n iterations is $\mathsf{poly}(\lambda, \log(p), \log(M))$. Finally, in each run of line 2, a single pass over the entire stream is sufficient, resulting in $O(n)$ passes over the stream for each party P_i.

Acknowledgements. Rex Fernando, Elaine Shi, and Pratik Soni were sponsored by the Algorand Centres of Excellence (ACE) Programme, the Defense Advanced Research Projects Agency under award number HR001120C0086, the Office of Naval Research under award number N000142212064, and the National Science Foundation under award numbers 2128519 and 2044679. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity. Ilan Komargodski is the incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University, supported in part by an Alon Young Faculty Fellowship, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643).

References

1. Kook Jin Ahn and Sudipto Guha: Access to data and number of iterations: dual primal algorithms for maximum matching under resource constraints. ACM Trans. Parallel Comput. (TOPC) **4**(4), 17 (2018)
2. Andoni, A., Nikolov, A., Onak, K., Yaroslavtsev, G.: Parallel algorithms for geometric graph problems. In: STOC 2014 (2014)
3. Andoni, A., Stein, C., Zhong, P.: Log diameter rounds algorithms for 2-vertex and 2-edge connectivity. arXiv preprint arXiv:1905.00850 (2019)
4. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. In: 33rd Annual Symposium on Foundations of Computer Science, FOCS, pp. 14–23 (1992)
5. Arun, A., Ganesh, C., Lokam, S.V., Mopuri, T., Sridhar, S.: Dew: a transparent constant-sized polynomial commitment scheme. In: Public Key Cryptography, pp. 542–571 (2023)
6. Assadi, S.: Simple round compression for parallel vertex cover. CoRR, abs/1709.04599 (2017)
7. Assadi, S., Bateni, M.H., Bernstein, A., Mirrokni, V., Stein, C.: Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. arXiv preprint arXiv:1711.03076 (2017)

8. Assadi, S., Khanna, S.: Randomized composable coresets for matching and vertex cover. In: Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 3–12. ACM (2017)
9. Assadi, S., Sun, X., Weinstein, O.: Massively parallel algorithms for finding well-connected components in sparse graphs. CoRR, abs/1805.02974 (2018)
10. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC, pp. 21–31 (1991)
11. Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Comput. Complex. **1**, 3–40 (1991)
12. Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and MapReduce. Proc. VLDB Endow. **5**(5), 454–465 (2012)
13. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. Proc. VLDB Endow. **5**(7), 622–633 (2012)
14. Bateni, M.H., Bhaskara, A., Lattanzi, S., Mirrokni, V.: Distributed balanced clustering via mapping coresets. In: Advances in Neural Information Processing Systems, pp. 2591–2599 (2014)
15. Behnezhad, S., Derakhshan, M., Hajiaghayi, M.T., Karp, R.M.: Massively parallel symmetry breaking on sparse graphs: MIS and maximal matching. CoRR, abs/1807.06701 (2018)
16. Behnezhad, S., Hajiaghayi, M.T., Harris, D.G.: Exponentially faster massively parallel maximal matching. arXiv preprint arXiv:1901.03744 (2019)
17. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-Solomon interactive oracle proofs of proximity. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 14:1–14:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018)
18. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4
19. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness, pp. 5:1–5:32 (2020)
20. Bick, A., Kol, G., Oshman, R.: Distributed zero-knowledge proofs over networks. In: SODA, pp. 2426–2458 (2022)
21. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: Theory of Cryptography, pp. 168–197 (2020)
22. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4
23. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. IACR Cryptol. ePrint Arch., p. 846 (2014)
24. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12
25. Bootle, J., Chiesa, A., Hu, Y., Orrú, M.: Gemini: elastic snarks for diverse environments. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology–EUROCRYPT 2022. LNCS, vol. 13276, pp. 427–457. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_15

26. Brandt, S., Fischer, M., Uitto, J.: Matching and MIS for uniformly sparse graphs in the low-memory MPC model. CoRR, abs/1807.05374 (2018)
27. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_24
28. Chang, Y.-J., Fischer, M., Ghaffari, M., Uitto, J., Zheng, Y.: The complexity of $(\Delta+1)$ coloring incongested clique, massively parallel computation, and centralized local computation. arXiv preprint arXiv:1808.08419 (2018)
29. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26
30. Chung, K.-M., Ho, K.-Y., Sun, X.: On the hardness of massively parallel computation. In: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA, pp. 153–162 (2020)
31. Czumaj, A., Łącki, J., Mądry, A., Mitrović, S., Onak, K., Sankowski, P.: Round compression for parallel matching algorithms. In: STOC (2018)
32. da Ponte Barbosa, R., Ene, A., Nguyen, H.L., Ward, J.: A new framework for distributed submodular maximization. In: FOCS, pp. 645–654 (2016)
33. Ene, A., Im, S., Moseley, B.: Fast clustering using MapReduce. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 681–689. ACM (2011)
34. Ene, A., Nguyen, H.: Random coordinate descent methods for minimizing decomposable submodular functions. In: International Conference on Machine Learning, pp. 787–795 (2015)
35. Fernando, R., Gelles, Y., Komargodski, I., Shi, E.: Maliciously secure massively parallel computation for all-but-one corruptions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology. CRYPTO 2022. LNCS, vol. 13507, pp. 688–718. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15802-5_24
36. Fernando, R., Komargodski, I., Liu, Y., Shi, E.: Secure massively parallel computation for dishonest majority. In: Theory of Cryptography - 18th International Conference, TCC, pp. 379–409 (2020)
37. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
38. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC, pp. 99–108 (2011)
39. Ghaffari, M., Lattanzi, S., Mitrović, S.: Improved parallel algorithms for density-based network clustering. In: International Conference on Machine Learning, pp. 2201–2210 (2019)
40. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. J. ACM **38**(3), 691–729 (1991)
41. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
42. Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 938–948 (2010)

43. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Constant-size commitments to polynomials and their applications. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
44. Kattis, A.A., Panarin, K., Vlasov, A.: Redshift: transparent snarks from list polynomial commitments. In: CCS, pp. 1725–1737 (2022)
45. Kol, G., Oshman, R., Saxena, R.R.: Interactive distributed proofs. In: PODC, pp. 255–264 (2018)
46. Kumar, R., Moseley, B., Vassilvitskii, S., Vattani, A.: Fast greedy algorithms in MapReduce and streaming. TOPC. **2**(3), 1–22 (2015)
47. Lee, J.: Dory: efficient, transparent arguments for generalised inner products and polynomial commitments. In: Theory of Cryptography, pp. 1–34 (2021)
48. Lindell: Parallel coin-tossing and constant-round secure two-party computation. J. Cryptol. **16**, 143–184 (2003)
49. Naor, M.: On cryptographic assumptions and challenges. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 96–109. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_6
50. Naor, M., Parter, M., Yogev, E.: The power of distributed verifiers in interactive proofs. In: SODA, pp. 1096–115 (2020)
51. Ozdemir, A., Boneh, D.: Experimenting with collaborative ZK-snarks: zero-knowledge proofs for distributed secrets. In: USENIX, pp. 4291–4308 (2022)
52. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Theory of Cryptography, pp. 222–242 (2013)
53. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM. **21**(2), 120–126 (1978)
54. Roughgarden, T., Vassilvitskii, S., Wang, J.R.: Shuffles and circuits (on lower bounds for modern parallel computation). J. ACM **65**(6), 1–24 (2018)
55. Setty, S., Lee, J.: Quarks: quadruple-efficient transparent Zksnarks. Cryptology ePrint Archive, Paper 2020/1275 (2020)
56. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient Zksnarks without trusted setup. In: S&P, pp. 926–943 (2018)
57. Wesolowski, B.: Efficient verifiable delay functions. J. Cryptol. **33**(4), 2113–2147 (2020)
58. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: a distributed zero knowledge proof system. In: USENIX, pp. 675–692 (2018)
59. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: S&P, pp. 859–876 (2020)