Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads

Alexander R. Block[1(✉)], Justin Holmgren[2], Alon Rosen[3], Ron D. Rothblum[4], and Pratik Soni[5]

[1] Purdue University, West Lafayette, USA
block9@purdue.edu
[2] NTT Research, East Palo Alto, USA
justin.holmgren@ntt-research.com
[3] IDC Herzliya, Herzliya, Israel
alon.rosen@idc.ac.il
[4] Technion, Haifa, Israel
rothblum@cs.technion.ac.il
[5] University of California, Santa Barbara, Santa Barbara, USA
pratik_soni@cs.ucsb.edu

Abstract. Zero-knowledge protocols enable the truth of a mathematical statement to be certified by a verifier without revealing any other information. Such protocols are a cornerstone of modern cryptography and recently are becoming more and more practical. However, a major bottleneck in deployment is the efficiency of the prover and, in particular, the space-efficiency of the protocol.

For every NP relation that can be verified in time T and space S, we construct a public-coin zero-knowledge argument in which the prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$. Our proofs have length $\text{polylog}(T)$ and the verifier runs in time $T \cdot \text{polylog}(T)$ (and space $\text{polylog}(T)$). Our scheme is in the random oracle model and relies on the hardness of discrete log in prime-order groups.

Our main technical contribution is a new space efficient *polynomial commitment scheme* for multi-linear polynomials. Recall that in such a scheme, a sender commits to a given multi-linear polynomial $P : \mathbb{F}^n \to \mathbb{F}$ so that later on it can prove to a receiver statements of the form "$P(x) = y$". In our scheme, which builds on commitments schemes of Bootle et al. (Eurocrypt 2016) and Bünz et al. (S&P 2018), we assume that the sender is given multi-pass streaming access to the evaluations of P on the Boolean hypercube and we show how to implement both the sender and receiver in roughly time 2^n and space n and with communication complexity roughly n.

1 Introduction

Zero-knowledge protocols are a cornerstone of modern cryptography, enabling the truth of a mathematical statement to be certified by a prover to a verifier

R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12551, pp. 168–197, 2020.
https://doi.org/10.1007/978-3-030-64378-2_7

without revealing any other information. First conceived by Goldwasser, Micali, and Rackoff [27], zero knowledge has myriad applications in both theory and practice and is a thriving research area today. Theoretical work primarily investigates the complexity tradeoffs inherent in zero-knowledge protocols:

- the number of rounds of interaction,
- the number of bits exchanged between the prover and verifier
- the computational complexity of the prover and verifier (e.g. running time, space usage)
- the degree of soundness—in particular, soundness can be statistical or computational, and the protocol may or may not be a proof of knowledge.

ZK-SNARKs (Zero-Knowledge Succinct Non-interactive ARguments of Knowledge) are protocols that achieve particularly appealing parameters: they are *non-interactive* protocols in which to certify an NP statement x with witness w, the prover sends a proof string π of length $|\pi| \ll |w|$. Such proof systems require setup (namely, a common reference string) and (under widely believed complexity-theoretic assumptions [24,25]) are limited to achieving computational soundness.

One of the main bottlenecks limiting the scalability of ZK-SNARKs is the high computational complexity of generating proof strings. In particular, a major problem is that even for the lowest-overhead ZK-SNARKs (see e.g. [4,22,39] and follow-up works), the prover requires $\Omega(T)$ *space* to certify correctness of a time-T computation, even if that computation uses space $S \ll T$.

As typical computations require much less space than time, such space usage can easily become a hard bottleneck. While it is straight-forward to run a program for as long as one's patience allows, a computer's memory cannot be expanded without purchasing additional hardware. Moreover, the memory architecture of modern computer systems is hierarchical, consisting of different tiers (various cache levels, RAM, and nonvolatile storage), with latencies and capacities that increase by orders of magnitude at each successive level. In other words, high space usage can also incur a heavy penalty in running time.

In this work, we focus on uniform non-deterministic computations—that is, proving that a nondeterministic time-T space-S Turing machine accepts an input x. Our objective is to obtain "complexity-preserving" (ZK-)SNARKs [10] for such computations, i.e., SNARKs in which the prover runs in time roughly T and space roughly S. Relatively efficient *privately verifiable* solutions are known [11,29]. In such schemes the verifier holds some secret state that, if leaked, compromises soundness. However, many applications (such as cryptocurrencies or other massively decentralized protocols) require public verifiability, which is the emphasis of our work.

To date, *publicly verifiable* complexity-preserving SNARKs are known only via recursive composition [9,47]. This approach indeed yields SNARKs with prover running time $\tilde{O}(T)$ and space usage $S \cdot \mathrm{polylog}(T)$, but with significant concrete overheads. Recursively composed SNARKs require both the prover and verifier to make non-black-box usage of an "inner" verifier for a different SNARK, leading to enormous computational overhead in practice.

Several recent works [14,16,18] attempt to solve the inefficiency problems with recursive composition, but the protocols in these works rely on heuristic and poorly understood assumptions to justify their soundness. While any SNARK (with a black-box security reduction) inherently relies on non-falsifiable assumptions [23], these SNARKs possess additional troubling features. They rely on hash functions that are modeled as random oracles in the security proof, despite being used in a non-black-box way by the honest parties. Security thus cannot be reduced to a simple computational hardness assumption, even in the random oracle model. Moreover, the practicality of the schemes crucially requires usage of a novel hash function (e.g., Rescue [1]) with algebraic structure designed to maximize the *efficiency* of non-black-box operations. Such hash functions have endured far less scrutiny than standard SHA hash functions, and the algebraic structure could potentially lead to a security vulnerability.

In this work, we ask:

Can we devise a complexity-preserving ZK-SNARK in the random oracle model based on standard cryptographic assumptions?

1.1 Our Results

Our main result is an affirmative answer to this question.

Theorem 1. *Assume that the discrete-log problem is hard in obliviously sampleable*[1] *prime-order groups. Then, for every* NP *relation that can be verified by a random access machine in time T and space S, there exists a publicly verifiable ZK-SNARK, in the random oracle model, in which both the prover and verifier run in time $T \cdot \mathrm{polylog}(T)$, the prover uses space $S \cdot \mathrm{polylog}(T)$, and the verifier uses space $\mathrm{polylog}(T)$. The proof length is poly-logarithmic in T.*

We emphasize that the verifier in our protocol has similar running time to that of the prover, in contrast to other schemes in the literature that offer *poly-logarithmic* time verification. While this limits the usefulness of our scheme in delegating (deterministic) computations, our scheme is well-geared towards *zero-knowledge* applications in which the prover and verifier are likely to have similar computational resources.

At the heart of our ZK-SNARK for NP relations verifiable by time-T space-S random access machine (RAM) is a new public-coin *interactive* argument of knowledge, in the random oracle model, for the same relation where the prover runs in time $T \cdot \mathrm{polylog}(T)$ and requires space $S \cdot \mathrm{polylog}(T)$. We make this argument zero-knowledge by using standard techniques which incurs minimal

[1] By *obliviously sampleable* we mean that there exist algorithms S and S^{-1} such that on input random coins r, the algorithm S samples a uniformly random group element g, whereas on input g, the algorithm S^{-1} samples random coins r that are consistent with the choice of g. In other words, if S uses ℓ random bits then the joint distributions $(U_\ell, S(U_\ell))$ and $(S^{-1}(S(U_\ell)), S(U_\ell))$ are identically distributed, where U_ℓ denotes the uniform distribution on ℓ bit strings..

asymptotic blow-up in the efficiency of the argument [2,20,48]. Finally, applying the Fiat-Shamir transformation [21] to our public-coin zero-knowledge argument yields Theorem 1.

Space-Efficient Polynomial Commitment for Multi-linear Polynomials. The key ingredient in our public-coin interactive argument of knowledge is a new space efficient *polynomial commitment scheme*, which we describe next.

Polynomial commitment schemes were introduced by Kate et al. [32] and have since received much attention [3,7,17,33,49,50], in particular due to their usage in the construction of efficient zero-knowledge arguments. Informally, a polynomial commitment scheme is a cryptographic primitive that allows a committer to send to a receiver a commitment to an n-variate polynomial $Q : \mathbb{F}^n \to \mathbb{F}$, over some finite field $\mathbb{F}$, and later reveal evaluations y of Q on a point $\mathbf{x} \in \mathbb{F}^n$ of the receiver's choice along with a proof that indeed $y = Q(\mathbf{x})$.

In this work we construct polynomial commitment schemes where the space complexity is (roughly) *logarithmic* in the description size of the polynomial. In order to state this result more precisely, we must first determine the type of access that the committer has to the polynomial.

We first note that in this work we restrict our attention to *multi-linear* polynomials (i.e., polynomials which have individual degree 1). Note that such a polynomial $Q : \mathbb{F}^n \to \mathbb{F}$ is uniquely determined by its evaluations on the Boolean hybercube, that is, $\big(Q(0), \ldots, Q(2^n - 1)\big)$, where the integers in $\mathbb{Z}_{2^n}$ are associated with vectors in $\{0,1\}^n$ in the natural way.

Towards achieving our space efficient implementation, and motivated by our application to the construction of an efficient argument-scheme, we assume that the committer has *multi-pass streaming access* to the evaluations of the polynomial on the Boolean hypercube. Such an access pattern can be modeled by giving the committer access to a read-only tape that is pre-initialized with the values $\big(Q(0), \ldots, Q(2^n - 1)\big)$. At every time-step the committer is allowed to either move the machine head to the right or to restart its position to 0.

Theorem 2 (Informal, see Theorem 5). *Let $\mathbb{G}$ be an obliviously sampleable group of prime-order p and let $Q : \mathbb{F}^n \to \mathbb{F}$ be some n-variate multi-linear polynomial. Assuming the hardness of discrete-log over $\mathbb{G}$ and multi-pass streaming access to the sequence $(Q(0), \ldots, Q(2^n - 1))$, there exists a polynomial commitment scheme for Q in the random oracle model such that*

1. *The commitment consists of one group element, evaluation proofs consist of $O(n)$ group and field elements,*
2. *The committer and receiver perform $\tilde{O}(2^n)$ group and field operations, make $\tilde{O}(2^n)$ queries to the random oracle, and store only $O(n)$ group and field elements, and*
3. *The committer makes $O(n)$ passes over $(Q(0), \ldots, Q(2^n - 1))$.*

Following [32], a number of works have focussed on achieving asymptotically optimal proof sizes (more generally, communication), and time complexity for both committer and receiver. However, the space complexity of the committer

has been largely ignored; naively it is lower-bounded by the size of the committer's input (which is a description of the polynomial). As mentioned above, we believe that obtaining a space-efficient polynomial commitment scheme in the streaming model to be of independent interest and may even eventually lead to significantly improved performance of interactive oracle proofs, SNARKS, and related primitives in practice.

We also mention that the streaming model is especially well-suited to our application of building space-efficient SNARKs. The reason is that in such schemes, the prover typically uses a polynomial commitment scheme to commit to a low-degree extension of the transcript of a RAM program, which, naturally, can be generated as a stream in space that is proportional to the space complexity of the underlying RAM program.

At a high level, we use an algebraic homomorphic commitment (e.g., Pedersen commitment [40]) to succinctly commit to the polynomial Q (by committing to the sequence $(Q(0), \ldots, Q(2^n - 1))$. Next, to provide evaluation proofs, our scheme leverages the fact that evaluating Q on point $\mathbf{x}$ reduces to computing an inner-product between $(Q(0), \ldots, Q(2^n - 1))$ and the sequence of Lagrange coefficients defined by the evaluation point $\mathbf{x}$. Relying on the homomorphic properties of our commitment, the basic step of our evaluation protocol is a 2-move (randomized) reduction step which allows the committer to "fold" a statement of size 2^n into a statement of size $2^n/2$. Our scheme is inspired from the "inner-product argument" of Bootle et al. [13] (and its variants [15,48]) but differs in the 2-move reduction step. More specifically, their reduction step folds the left half of $(Q(0), \ldots, Q(2^n - 1))$ with its right half (referred to as msb-based folding as the index of the elements that are folded differ in the most significant bit). This, unfortunately, is not compatible with our streaming model (we explain this shortly). We instead perform the more natural *lsb-based folding* which, indeed, is compatible with the streaming model. We additionally exploit random access to the inner-product argument's setup parameters (defined by the random oracle) and the fact that any component of the coefficient sequence can be computed in polylogarithmic time, i.e. $\mathsf{poly}(n)$ time. We give a high level overview of our scheme in Sect. 2.1.

1.2 Prior Work

Complexity Preserving ZK-SNARKs. Bitansky and Chiesa [11] proposed to construct complexity preserving ZK-SNARKS by first constructing complexity preserving multi-prover interactive proof (MIPs) and then compile them using cryptographic techniques. While our techniques share the same high-level approach, our compilation with a polynomial-commitment scheme yields a publicly verifiable scheme whereas [11] only obtain a designated verifier scheme.

Blumberg et al. [12] give a 2-prover complexity preserving MIP of knowledge, improving (concretely) on the complexity preserving MIP of [11] (who obtain a 2-prover MIP via a reduction from their many-prover MIP). Both Bitansky and Chiesa and Blumberg et al. obtain their MIPs from reducing RAMs to circuits via the reduction of Ben-Sasson et al. [5], then appropriately arithmetize the circuit

into an algebraic constraint satisfaction problem. Holmgren and Rothblum [29] obtain a non-interactive protocol based on standard (falsifiable assumptions) by also constructing a complexity preserving MIP for RAMs (achieving no-signaling soundness) and compiling it into an argument using fully-homomorphic encryption (á la [8,30,31]). We remark that [29] reduce a RAM directly to algebraic constraints via a different encoding of the RAM transcript, thereby avoiding the reduction to circuits entirely.

Another direction for obtaining complexity preserving ZK-SNARKS is via recursive composition [9,47], or "bootstrapping". Here, one begins with an "inefficient" SNARK and bootstraps it recursively to produce publicly verifiable complexity preserving SNARKs. While these constructions yield good asymptotics, these approaches require running the inefficient SNARK on many sub-computations. Recent works [14,16,18] describe a novel approach to recursive composition which attempt to solve the inefficiencies of the aforementioned recursive compositions, though at a cost to the theoretical basis for the soundness of their scheme (as discussed above).

Interactive Oracle Proofs. Interactive oracle proofs (IOPs), introduced by Ben-Sasson et al. [6] and independently by Reingold et al. [41], are interactive protocols where a verifier has oracle access to all prover messages. IOPs capture (and generalize), both interactive proofs and PCPs.

A recent line of work [5,12,19,26,42,44,46,48] follows the framework of Kilian [34] and Micali [37] to obtain efficient arguments by constructing efficient IOPs and compiling them into interactive arguments using collision resistant hashing [6,34] or the random oracle model [6,37].

Polynomial Commitments. Polynomial commitment schemes were introduced by Kate et al. [32] and have since been an active area of research. Lines of research for construction polynomial commitment schemes include privately verifiable schemes [32,38], publicly-verifiable schemes with trusted setup [17], and zero-knowledge schemes [49]. More recently, much focus has been on obtaining publicly-verifiable schemes without a trusted setup [3,7,17,33,49,50]. We note that in all prior works on polynomial commitments, the space complexity of the sender is proportional to the description size of the polynomial, whereas we achieve *poly-logarithmic* space complexity.

2 Technical Overview

As mentioned above, the key component in our construction is that of a public-coin interactive argument for RAM computations. The latter construction itself consists of two key technical ingredients. First, we construct a *polynomial interactive oracle proof* (polynomial IOP) for time-T space-S RAM computations in which the prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$. We note that this ingredient is a conceptual contribution which formalizes prior work in the language of polynomial IOPs. Second, we compile this IOP with

a space-efficient extractable *polynomial commitment scheme* where the prover has multi-pass streaming access to the polynomial to which it is committing—a property that plays nicely with the streaming nature of RAM computations. We emphasize that the construction of the space-efficient polynomial commitment scheme is our main technical contribution, and describe our scheme in more detail next.

2.1 Polynomial Commitment to Multi-linear Polynomials in the Streaming Model

Fix a finite field $\mathbb{F}$ of prime order p. Also fix an obliviously sampleable (see Footnote 1) group $\mathbb{G}$ of order p in which the discrete logarithm is hard. Let $H : \{0,1\}^* \to \mathbb{G}$ be the random oracle.

In order to describe our polynomial commitment scheme, we start with some notation. Let n be a positive integer and set $N = 2^n$. We will be considering N-dimensional vectors over $\mathbb{F}$ and will index such vectors using n dimensional binary vectors. For example, if $\mathbf{b} \in \mathbb{F}^{2^6}$ then $\mathbf{b}_{000101} = b_5$. For convenience, we will denote $\mathbf{b} \in \mathbb{F}^N$ by $(b_{\mathbf{c}} : \mathbf{c} \in \{0,1\}^n)$ where $b_{\mathbf{c}}$ is the $\mathbf{c}$-th element of $\mathbf{b}$. For $\mathbf{b} = (b_n, \ldots, b_1) \in \{0,1\}^n$ we refer to b_1 as the least-significant bit (lsb) of $\mathbf{b}$. Finally, for $\mathbf{b} \in \mathbb{F}^N$, we denote by $\mathbf{b}_e$ the restriction of $\mathbf{b}$ to the even indices, that is, $\mathbf{b}_e = (b_{\mathbf{c}0} : \mathbf{c} \in \{0,1\}^{n-1})$. Similarly, we denote by $\mathbf{b}_o = (b_{\mathbf{c}1} : \mathbf{c} \in \{0,1\}^{n-1})$ the restriction of $\mathbf{b}$ to odd indices.

Let $Q : \mathbb{F}^n \to \mathbb{F}$ be a multi-linear polynomial. Recall that such a polynomial can be fully described by the sequence of its evaluations over the Boolean hypercube. More specifically, for any $\mathbf{x} \in \mathbb{F}^n$, the evaluation of Q on $\mathbf{x}$ can be expressed as

$$Q(\mathbf{x}) = \sum_{\mathbf{b} \in \{0,1\}^n} Q(\mathbf{b}) \cdot z(\mathbf{x}, \mathbf{b}), \tag{1}$$

where $z(\mathbf{x}, \mathbf{b}) = \prod_{i \in [n]} \big(b_i \cdot x_i + (1 - b_i) \cdot (1 - x_i)\big)$. We use $\mathbf{Q} \in \mathbb{F}^N$ to denote the restriction of Q to the Boolean hybercube (i.e., $\mathbf{Q} = (Q(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n)$).

Next, we describe the our commitment scheme which has three phases: (a) Setup, (b) Commit and (c) Evaluation.

Setup and Commit Phase. During setup, the committer and receiver both consistently define a sequence of N generators for $\mathbb{G}$ using the random oracle, that is, $\mathbf{g} = (g_{\mathbf{b}} = H(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n)$. Then, given streaming access to $\mathbf{Q}$, the committer computes the Pedersen multi-commitment [40] C defined as

$$C = \prod_{\mathbf{b} \in \{0,1\}^n} (g_{\mathbf{b}})^{Q_{\mathbf{b}}}. \tag{2}$$

For $\mathbf{g} \in \mathbb{G}^{2^n}$ and $\mathbf{Q} \in \mathbb{F}^{2^n}$, we use $\mathbf{g}^{\mathbf{Q}}$ as a shorthand to denote the value $\prod_{\mathbf{b} \in \{0,1\}^n} (g_{\mathbf{b}})^{Q_{\mathbf{b}}}$. Assuming the hardness of discrete-log for $\mathbb{G}$, we note that C in Eq. (2) is a binding commitment to $\mathbf{Q}$ under generators $\mathbf{g}$. Note that the

committer only needs to perform a single-pass over $\mathbf{Q}$ and performs N exponentiations to compute C while storing only $O(1)$ number of group and field elements.[2]

Evaluation Phase. On input an evaluation point $\mathbf{x} \in \mathbb{F}^n$, the committer computes and sends $y = Q(\mathbf{x})$ and defines the auxiliary commitment $C_y \leftarrow C \cdot g^y$ for some receiver chosen generator g. Then, both engage in an argument (of knowledge) for the following NP statement which we refer to as the "inner-product" statement:

$$\exists \mathbf{Q} \in \mathbb{Z}_p^N \ : \ y = \langle \mathbf{Q}, \mathbf{z} \rangle \text{ and } \ C_y = g^y \cdot \mathbf{g}^{\mathbf{Q}} \ , \tag{3}$$

where $\mathbf{z} = (z(\mathbf{x}, \mathbf{b}) : \mathbf{b} \in \{0,1\}^n)$ as defined in Eq. (1). This step can be viewed as proving knowledge of the decommitment $\mathbf{Q}$ of the commitment C_y, which furthermore is consistent with the inner-product claim that $y = \langle \mathbf{Q}, \mathbf{z} \rangle$.

Inner-product Argument. A basic step in the argument for the above inner-product statement is a 2-move randomized reduction step which allows the prover to decompose the N-sized statement $(C_y, \mathbf{z}, y)$ into two $N/2$-sized statements and then "fold" them into a single $N/2$-sized statement $(\bar{C}_{\bar{y}}, \bar{\mathbf{z}} = (\bar{z}_{\mathbf{c}} : \mathbf{c} \in \{0,1\}^{n-1}), \bar{y})$ using the verifier's random challenge. We explain the two steps below (as well as in Fig. 1).

1. Committer computes the cross-product $y_e = \langle \mathbf{Q}_e, \mathbf{z}_o \rangle$ between the even-indexed elements $\mathbf{Q}_e$ with the odd-indexed vectors $\mathbf{z}_o$. Furthermore, it computes a binding commitment C_e that binds y_e (with g) and $\mathbf{Q}_e$ (with $\mathbf{g}_o$). That is,
$$C_e = g^{y_e} \cdot \mathbf{g}_o^{\mathbf{Q}_e} \ , \tag{4}$$
where recall that for $\mathbf{g} = (g_1, \ldots, g_t)$ and $\mathbf{x} = (x_1, \ldots, x_t)$ the expression $\mathbf{g}^{\mathbf{x}} = \prod_{i \in [t]} g_i^{x_i}$. This results in an $N/2$-sized statement $(C_e, \mathbf{z}_o, y_e)$ with witness $\mathbf{Q}_e$. Similarly, as in Fig. 1 it computes the second $N/2$-sized statement $(C_o, \mathbf{z}_e, y_o)$ with witness $\mathbf{Q}_o$. The committer sends (y_e, y_o, C_e, C_o) to the receiver.
2. After receiving a random challenge $\alpha \in \mathbb{F}^*$, committer folds its witness $\mathbf{Q}$ into an $N/2$-sized vector $\bar{\mathbf{Q}} = \alpha \cdot \mathbf{Q}_e + \alpha^{-1} \cdot \mathbf{Q}_o$. More specifically, for every $\mathbf{c} \in \{0,1\}^{n-1}$,
$$\bar{Q}_{\mathbf{c}} = \alpha \cdot Q_{\mathbf{c}0} + \alpha^{-1} \cdot Q_{\mathbf{c}1} \ . \tag{5}$$
Similarly, the committer and receiver both compute the rest of the folded statement $(\bar{C}_{\bar{y}}, \bar{\mathbf{z}}, \bar{y})$ as shown in Fig. 1.

Relying on the homomorphic properties of Pedersen commitments, it can be shown that if $\mathbf{Q}$ were a witness to $(C_y, \mathbf{z}, y)$ then $\bar{\mathbf{Q}}$ is a witness for $(\bar{C}_{\bar{y}}, \bar{\mathbf{z}}, \bar{y})$.[3] In the actual protocol, the parties then recurse on smaller statements $(\bar{C}_{\bar{y}}, \bar{\mathbf{z}}, \bar{y})$

[2] Here, we treat exponentiation as an atomic operation but note that computing g^α for $\alpha \in \mathbb{Z}_p$ can be emulated, via repeated squarings, by $O(\log p)$ group multiplications while storing only $O(1)$ number of group and field elements.

[3] Albeit under different set of generators but we ignore this for now.

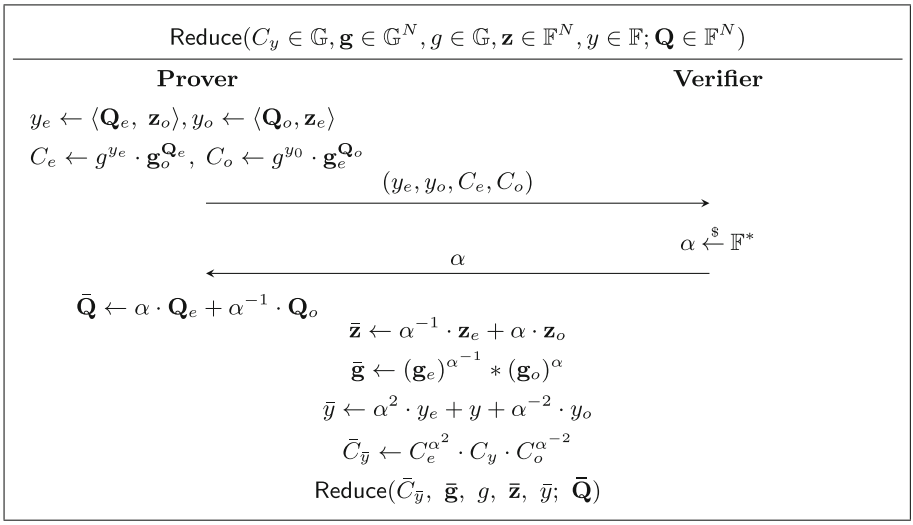

Fig. 1. Our 2-move randomized reduction step for the inner-product protocol where recall that for any $\mathbf{Q} \in \mathbb{F}^N$, we denote by $\mathbf{Q}_e$ the elements of $\mathbf{Q}$ indexed by even numbers where $\mathbf{Q}_o$ denotes the elements with odd indices. On input a statement of size $N > 1$, Reduce results in a statement of size $N/2$.

forming a recursion tree. After $\log N$ steps, the statement is of size 1 in which case the committer sends its witness which is a single field element. This gives an overall communication of $O(\log N)$ field and group elements. Next we briefly discuss the efficiency of the scheme.

Efficiency. For the purpose of this overview, we focus only on the time and space efficiency of the committer in the inner-product argument (the analysis for the receiver is analogous). Recall that in a particular step of the recursion, suppose we are recursing on the $N/2$-sized statement $(\bar{C}_{\bar{y}}, \bar{\mathbf{z}}, \bar{y})$ with witness $\bar{\mathbf{Q}}$, the committer's computation includes computing (a) the cross-product $\langle \bar{\mathbf{Q}}_e, \bar{\mathbf{z}}_o \rangle$ between the even half of $\bar{\mathbf{Q}}$ and the odd half of $\bar{\mathbf{z}}$, and (b) the "cross-exponentiation" $\bar{\mathbf{g}}_o^{\bar{\mathbf{Q}}_e}$ of the even half of $\bar{\mathbf{Q}}$ with the odd half of the generators $\bar{\mathbf{g}}$.[4]

A straightforward approach to compute (a) is to have $\bar{\mathbf{Q}}$ (and $\bar{\mathbf{z}}$) in memory, but this requires the committer to have $\Omega(N)$ space which we want to avoid. Towards a space efficient implementation, first note every element of $\bar{\mathbf{Q}}$ depends on only two, more importantly, consecutive elements of $\mathbf{Q}$. This coupled with streaming access to $\mathbf{Q}$ is sufficient to simulate streaming access to $\bar{\mathbf{Q}}$ while making only **one** pass over $\mathbf{Q}$. Secondly, by definition, computing any element of $\mathbf{z}$ requires only $O(\log N)$ field operations while storing only $O(n)$ field elements This then allows to compute any element of $\bar{\mathbf{z}}$ on the fly with polylog(N) operations. Given the simulated streaming access to $\bar{\mathbf{Q}}$ along with

[4] Efficiency for $\langle \bar{\mathbf{Q}}_o, \bar{\mathbf{z}}_e \rangle$ and $\bar{\mathbf{g}}_e^{\bar{\mathbf{Q}}_o}$ can be argued similarly.

Scheme	[13,15] (msb-based)	This work (lsb-based)
$\bar{Q}_{\mathbf{b}}$	$\alpha \cdot Q_{0\mathbf{b}} + \alpha^{-1} \cdot Q_{1\mathbf{b}}$	$\alpha \cdot Q_{\mathbf{b}0} + \alpha^{-1} \cdot Q_{\mathbf{b}1}$
$\bar{z}_{\mathbf{b}}$	$\alpha^{-1} \cdot z_{0\mathbf{b}} + \alpha \cdot z_{1\mathbf{b}}$	$\alpha^{-1} \cdot z_{\mathbf{b}0} + \alpha \cdot z_{\mathbf{b}1}$
$\bar{g}_{\mathbf{b}}$	$(g_{0\mathbf{b}})^{\alpha^{-1}} * (g_{1\mathbf{b}})^{\alpha}$	$(g_{\mathbf{b}0})^{\alpha^{-1}} * (g_{\mathbf{b}1})^{\alpha}$

Fig. 2. Table highlights the differences between the 2-move randomized reduction steps of the inner-product argument of [13,15] (second column) and our scheme (third column). Specifically, given $\mathbf{Q}, \mathbf{z}, \mathbf{g}$ of size 2^n, the rows describe the definition of the $2^n/2$ sized vectors $\bar{\mathbf{Q}}, \bar{\mathbf{z}}, \bar{\mathbf{g}}$ respectively where $\mathbf{b} \in \{0,1\}^{n-1}$.

the ability to compute any element of $\bar{\mathbf{z}}$ on the fly is sufficient to compute the $\langle \bar{\mathbf{Q}}_e, \bar{\mathbf{z}}_o \rangle$. Note this step, overall, requires performing only a single pass over $\mathbf{Q}$ and $N \cdot \text{polylog} N$ operations, and storing only the evaluation point $\mathbf{x}$ and verifier challenge α (along with some book-keeping). The computation of (b) is handled similarly, except that here we crucially leverage the fact that $\mathbf{g}$ is defined using the random oracle, and hence the committer has random access to all of the generators in $\mathbf{g}$. Relying on similar ideas as in (a), the committer can compute $\bar{\mathbf{g}}_o^{\bar{\mathbf{Q}}_e}$ while additionally making $O(N)$ queries to the random oracle. Overall, this gives the required prover efficiency. Please see Sect. 4.3 for a full discussion on the efficiency.

Comparison with the 2-move Reduction Step of [13,48]. In their protocol, a major difference is in how the folding is performed (Step 2, Fig. 1). We list concrete differences in Fig. 2. But at a high level, since they fold the first element $Q_{00^{n-1}}$ with the $N/2$-nd element $Q_{10^{n-1}}$, it takes at least a one pass over $\mathbf{Q}$ to even compute the first element of $\bar{\mathbf{Q}}$, thereby requiring $\Omega(N)$ passes over $\mathbf{Q}$ which is undesirable.[5] Although we differ in the 2-move reduction steps, the security of our scheme follows from ideas similar to [13,48].

2.2 Polynomial IOPs for RAM Programs

The second ingredient we use to obtain space-efficient interactive arguments for NP relations verifiable by time-T space-S RAMs is a space-efficient *polynomial interactive oracle proof system* [6,17,41]. Informally, an interactive oracle proof (IOP) is an interactive protocol such that in each round the verifier sends a message to the prover, and the prover responds with proof string that the verifier can query in only a few locations. A polynomial IOP is an IOP where the proof string sent by the prover is a polynomial (i.e, all evaluations of a polynomial on a domain), and if a cheating prover successfully convinces a verifier then the proof string is consistent with some polynomial.

[5] When a polynomial commitment is used in building arguments, it takes $O(N)$ time to stream $\mathbf{Q}$, and requiring $\Omega(N)$ passes results in a prover that runs in quadratic time.

We consider a variant of the polynomial IOP model in which the prover sends messages which are encoded by the channel; in particular, the time and space complexity of the encoding computed by the channel do not factor into the complexity of the prover. For our purposes, we use the polynomial IOP that is implicit in [12] and consider it with a channel which computes multi-linear extensions of the prover messages. We briefly describe the IOP construction for completeness (see Sect. 5 for more details). The polynomial IOP at its core first leverages the space-efficient RAM to arithmetic circuit satisfiability reduction of [12] (adapting techniques of [5]). This reduction transforms a time-T space-S RAM into a circuit of size $T \cdot \text{polylog}(T)$ and has the desirable property (for our purposes) that the circuit can be accessed by the prover in a streaming manner: the assignment of gate values in the circuit can be streamed "gate-by-gate" in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$, which, in particular, allows a prover to compute a correct transcript of the circuit in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$.

The prover sends the verifier an oracle that is the multi-linear extension of the gate values (i.e., the transcript), where we remark that this extension is computed by the channel. The correctness of the computation is reduced to an algebraic claim about a low degree polynomial which is identically 0 on the Boolean hypercube if and only if the circuit is satisfied by the given witness. Finally, the prover and verifier engage in the classical sum-check protocol [36, 45] to verify that the constructed polynomial indeed vanishes on the Boolean hypercube.

Theorem 3. *There exists a public-coin polynomial* IOP *over a channel which encodes prover messages as multi-linear extensions for* NP *relations verifiable by a time-T space-S random access machine M such that if $y = M(x; w)$ then*

1. *The* IOP *has perfect completeness and statistical soundness, and has $O(\log(T))$ rounds;*
2. *The prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$ (not including the space required for the oracle) when given input-witness pair $(x; w)$ for M, sends a single polynomial oracle in the first round, and has $\text{polylog}(T)$ communication in all subsequent rounds; and*
3. *The verifier runs in time $(|x| + |y|) \cdot \text{polylog}(T)$, space $\text{polylog}(T)$, and has query complexity 3.*

2.3 Obtaining Space-Efficient Interactive Arguments

We compile Theorem 3 and Theorem 2 into a space-efficient interactive argument scheme for NP relations verifiable by RAM computations.

Theorem 4 (Informal, see Theorem 6). *There exists a public-coin interactive argument for* NP *relations verifiable by a time-T space-S random access machine M, in the random oracle model, under the hardness of discrete-log in obliviously sampleable prime-order groups such that:*

1. *The prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$;*

2. *The verifier runs in time* $T \cdot \text{polylog}(T)$ *and space* $\text{polylog}(T)$*; and*
3. *The round complexity is* $O(\log T)$ *and the communication complexity is* $\text{polylog}(T)$.

The interactive argument of Theorem 4 is obtained by modifying the polynomial IOP of Theorem 3 with the commitment scheme of Theorem 2 in the following manner. First, the prover uses the polynomial commitment scheme to send a commitment to the multi-linear extension of the gate values rather than an oracle. This is possible to do in a space-efficient manner because of the streaming nature of RAM computations and the streaming nature of the IOP. Second, the verifier oracle querie are replaced with the prover and verifier engaging in the evaluation protocol of the polynomial commitment scheme. The remainder of the IOP protocol remains unchanged. Thus we obtain Theorem 4. We obtain Theorem 1 by transforming the interactive argument to a zero-knowledge interactive argument using standard techniques, then apply the Fiat-Shamir transformation [21].

3 Preliminaries

We let λ denote the security parameter, let $n \in \mathbb{N}$ and $N = 2^n$. For a finite, non-empty set S, we let $x \xleftarrow{\$} S$ denote sampling element x from S uniformly at random. We let $\mathsf{Primes}(1^\lambda)$ denote the set of all λ-bit primes. We let $\mathbb{F}_p$ denote a finite field of prime cardinality p, often use lower-case Greek letters to denote elements of $\mathbb{F}$, e.g., $\alpha \in \mathbb{F}$. For a group $\mathbb{G}$, we denote elements of $\mathbb{G}$ with sans-serif font; e.g., $\mathsf{g} \in \mathbb{G}$. We use boldface lowercase letters to denote binary vectors, e.g. $\mathbf{b} \in \{0,1\}^n$. We assume for a bit string $(b_n, \ldots, b_1) = \mathbf{b} \in \{0,1\}^n$ that b_n is the most significant bit and b_1 is the least significant bit. For bit string $\mathbf{b} \in \{0,1\}^n$ and $b \in \{0,1\}$ we let $b\mathbf{b}$ (resp., $\mathbf{b}b$) denote the string $(b \circ \mathbf{b}) \in \{0,1\}^{n+1}$ (resp.,$(\mathbf{b} \circ b) \in \{0,1\}^{n+1}$), where "$\circ$" is the string concatenation operator. We use boldface lowercase Greek denotes $\mathbb{F}$ vectors, e.g., $\boldsymbol{\alpha} \in \mathbb{F}^n$, and let $\boldsymbol{\alpha} = (\alpha_n, \ldots, \alpha_1)$ for $\alpha_i \in \mathbb{F}$. We let uppercase letters denote sequences and let corresponding lowercase letters to denote its elements, e.g., $Y = (y_{\mathbf{b}} \in \mathbb{F} : \mathbf{b} \in \{0,1\}^n)$ is a sequence of 2^n elements in $\mathbb{F}$. We denote by $\mathbb{F}^N$ the set of all sequences over $\mathbb{F}$ of size N.

Random Oracle. We let $\mathcal{U}(\lambda)$ denote the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. A random oracle with security parameter λ is a function $H : \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

3.1 The Discrete-Log Relation Assumption

Let GGen be an algorithm that on input $1^\lambda \in \mathbb{N}$ returns $(\mathbb{G}, p, \mathsf{g})$ such that $\mathbb{G}$ is the description of a finite cyclic group of prime order p, where p has length λ, and g is a generator of $\mathbb{G}$.

Assumption 1 (Discrete-log Assumption). *The Discrete-log Assumption holds for* GGen *if for all PPT adversaries* A *there exists a negligible function* $\mu(\lambda)$ *such that*

$$\Pr\left[\alpha' = \alpha : (\mathbb{G}, \mathsf{g}, p) \xleftarrow{\$} \mathsf{GGen}(1^\lambda),\ \alpha \xleftarrow{\$} \mathbb{Z}_p, \alpha' \xleftarrow{\$} A(\mathbb{G}, \mathsf{g}, \mathsf{g}^\alpha)\right] \leq \mu(\lambda)\ .$$

For our purposes, we use the following variant of the discrete-log assumption which is equivalent to Assumption 1.

Assumption 2 (Discrete-log Relation Assumption [13]). *The Discrete-log Relation Assumption holds for* GGen *if for all PPT adversaries* A *and for all* $n \geq 2$ *there exists a negligible function* $\mu(\lambda)$ *such that*

$$\Pr\left[\exists \alpha_i \neq 0 \wedge \prod_{i=1}^{n} \mathsf{g}_i^{\alpha_i} = 1 : \begin{array}{c}(\mathbb{G}, \mathsf{g}, p) \xleftarrow{\$} \mathsf{GGen}(1^\lambda),\ \mathsf{g}_1, \ldots, \mathsf{g}_n \xleftarrow{\$} \mathbb{G}\ , \\ (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_p^n \xleftarrow{\$} A(\mathbb{G}, \mathsf{g}_1, \ldots, \mathsf{g}_n)\ .\end{array}\right] \leq \mu(\lambda)\ .$$

We say $\prod_{i=1}^{n} \mathsf{g}_i^{\alpha_i} = 1$ is a non-trivial discrete log relation between $\mathsf{g}_1, \ldots, \mathsf{g}_n$. The Discrete Log Relation assumption states that an adversary can't find a non-trivial relation between randomly chosen group elements.

3.2 Interactive Arguments of Knowledge in ROM

Definition 1 (Witness Relation Ensemble). *A* witness relation ensemble *or relation ensemble is a ternary relation* $\mathcal{R}_L$ *that is polynomially bounded, polynomial time recognizable and defines a language* $\mathcal{L} = \{(pp, x) : \exists w \text{ s.t. } (pp, x, w) \in \mathcal{R}_\mathcal{L}\}$. *We omit pp when considering languages recognized by binary relations.*

Definition 2 (Interactive Arguments [27]). *Let* $\mathcal{R}$ *be some relation ensemble. Let* (P, V) *denote a pair of PPT interactive algorithms and* Setup *denote a non-interactive setup algorithm that outputs public parameters pp given security parameter* 1^λ. *Let* $\langle P(pp, x, w), V(pp, x)\rangle$ *denote the output of* V*'s interaction with* P *on common inputs public parameter pp and statement* x *where additionally* P *has the witness* w. *The triple* (Setup, P, V) *is an* argument for $\mathcal{R}$ *in the random oracle model (ROM) if*

1. *Perfect Completeness. For any adversary* A

$$\Pr\left[(x, w) \notin \mathcal{R} \text{ or } \langle P^H(pp, x, w), V^H(pp, x)\rangle = 1\right] = 1\ ,$$

where probability is taken over $H \xleftarrow{\$} \mathcal{U}(\lambda), pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda), (x, w) \xleftarrow{\$} A^H(pp)$.

2. *Computational Soundness. For any non-uniform PPT adversary* A

$$\Pr\left[\forall w\ (x, w) \notin \mathcal{R} \text{ and } \langle A^H(pp, x, st), V^H(pp, x)\rangle = 1\right] \leq \mathsf{negl}(\lambda)\ ,$$

where probability is taken over $H \xleftarrow{\$} \mathcal{U}(\lambda), pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda), (x, st) \xleftarrow{\$} A^H(pp)$.

Remark 1. Usually completeness is required to hold for all $(x, w) \in \mathcal{R}$. However, for the argument systems used in this work, statements x depends on pp output by Setup and the random oracle H. We model this by asking for completeness to hold for statements sampled by an adversary A, that is, for $(x, w) \xleftarrow{\$} A(pp)$.

For our applications, we will need (Setup, P, V) to be an argument of knowledge. Informally, in an argument of knowledge for $\mathcal{R}$, the prover convinces the verifier that it "knows" a witness w for x such that $(x, w) \in \mathcal{R}$. In this paper, knowledge means that the argument has *witness-extended emulation* [28,35].

Definition 3 (Witness-Extended Emulation). *Given a public-coin interactive argument tuple (Setup, P, V) and some arbitrary prover algorithm P^*, let $\mathsf{Record}(P^*, pp, x, st)$ denote the message transcript between P^* and V on shared input x, initial prover state st, and pp generated by Setup. Furthermore, let $\mathsf{E}^{\mathsf{Record}(P^*, pp, x, st)}$ denote a machine E with a transcript oracle for this interaction that can be rewound to any round and run again on fresh verifier randomness. The tuple (Setup, P, V) has witness-extended emulation if for every deterministic polynomial-time P^* there exists an expected polynomial-time emulator E such that for all non-uniform polynomial-time adversaries A the following holds:*

$$\Pr\left[A^H(tr) = 1 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda), \\ (x, st) \xleftarrow{\$} A^H(pp),\ tr \xleftarrow{\$} \mathsf{Record}^H(P^*, pp, x, st) \end{array}\right]$$

$$\approx \Pr\left[\begin{array}{c} A^H(tr) = 1 \text{ and} \\ tr \text{ accepting} \implies (x, w) \in \mathcal{R} \end{array} : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda), \\ (x, st) \xleftarrow{\$} A^H(pp), \\ (tr, w) \xleftarrow{\$} \mathsf{E}^{H, \mathsf{Record}^H(P^*, pp, x, st)}(pp, x) \end{array}\right]$$

It was shown in [13,17] that witness-extended emulation is implied by an extractor that can extract the witness given a tree of accepting transcripts. For completeness we state this—dubbed Generalized Forking Lemma—more formally below but refer to [17] for the proof.

Definition 4 (Tree of Accepting Transcripts). *An $(n_1, \ldots, n_r)$-tree of accepting transcripts for an interactive argument on input x is defined as follows: The root of the tree is labelled with the statement x. The tree has r depth. Each node at depth $i < r$ has n_i children, and each child is labeled with a distinct value for the i-th challenge. An edge from a parent node to a child node is labeled with a message from P to V. Every path from the root to a leaf corresponds to an accepting transcript, hence there are $\prod_{i=1}^{r} n_i$ distinct accepting transcripts overall.*

Lemma 1 (Generalized Forking Lemma [13,17]). *Let (Setup, P, V) be an r-round public-coin interactive argument system for a relation $\mathcal{R}$. Let T be a tree-finder algorithm that, given access to a $\mathsf{Record}(\cdot)$ oracle with rewinding capability, runs in polynomial time and outputs an $(n_1, \ldots, n_r)$-tree of accepting transcripts with overwhelming probability. Let Ext be a deterministic polynomial-time extractor algorithm that, given access to T's output, outputs a witness w for the statement x with overwhelming probability over the coins of T. Then, (P, V) has witness-extended emulation.*

Definition 5 (Public-coin). *An argument of knowledge is called public-coin if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages, i.e., the challenges correspond to the verifier's randomness H.*

Zero-Knowledge. We also need our argument of knowledge to be zero-knowledge, that is, to not leak partial information about w apart from what can be deduced from $(x, w) \in \mathcal{R}$.

Definition 6 (Zero-knowledge Arguments). *Let* (Setup, P, V) *be an public-coin interactive argument system for witness relation ensemble* $\mathcal{R}$*. Then,* (Setup, P, V) *has computational zero-knowledge with respect to an auxiliary input if for every PPT interactive machine* V^**, there exists a PPT algorithm* S*, called the simulator, running in time polynomial in the length of its first input, such that for every* $(x, w) \in \mathcal{R}$ *and any* $z \in \{0,1\}^*$*:*

$$View(\langle P(w), V^*(z)\rangle(x)) \approx_c S(x, z),$$

where $View(\langle P(w), V^*(z)\rangle(x))$ *denotes the distribution of the transcript of interaction between* P *and* V^**, and* $\approx_c$ *denotes that the two quantities are computationally indistinguishable. If the statistical distance between the two distributions is negligible then the interactive argument is said to be statistical zero-knowledge. If the simulatro is allowed to abort with probability at most* $1/2$*, but the distribution of its output conditioned on not aborting is identically distributed to* $View(\langle P(w), V^*(z)\rangle(x))$*, then the interactive argument is called perfect zero-knowledge.*

3.3 Multi-linear Extensions

Definition 7 (Multi-linear Extensions). *Let* $n \in \mathbb{N}$*,* $\mathbb{F}$ *be some finite field and let* $W : \{0,1\}^n \to \mathbb{F}$*. Then, the multi-linear extension of* W *(denoted as* $\mathsf{MLE}(W, \cdot) : \mathbb{F}^n \to \mathbb{F}$*) is the (unique) multi-linear polynomial that agrees with* W *on* $\{0,1\}^n$*. Equivalently,*

$$\mathsf{MLE}(W, \boldsymbol{\zeta} \in \mathbb{F}^n) = \sum_{\mathbf{b} \in \{0,1\}^n} W(\mathbf{b}) \cdot \prod_{i=1}^{n} \beta(b_i, \zeta_i) \ ,$$

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

For notational convenience, we denote $\prod_{i=1}^{k} \beta(b_i, \zeta_i)$ by $\overline{\beta}(\mathbf{b}, \boldsymbol{\zeta})$.

Remark 2. There is a bijective mapping between the set of all functions from $\{0,1\}^n \to \mathbb{F}$ to the set of all n-variate multi-linear polynomials over $\mathbb{F}$. More specifically, as seen above every function $W : \{0,1\}^n \to \mathbb{F}$ defines a (unique) multi-linear polynomial. Furthermore, every multi-linear polynomial $Q : \mathbb{F}^n \to \mathbb{F}$ is, in fact, the multi-linear extension of the function that maps $\mathbf{b} \in \{0,1\}^n \to Q(\mathbf{b})$.

Streaming Access to Multi-linear Polynomials. For our commitment scheme, we assume that the committer will have *multi-pass streaming* access to the function table of W (which defines the multi-linear polynomial) in the lexicographic ordering. Specifically, the committer will be given access to a read-only tape that is pre-initialized with the sequence $W = \left(w_{\mathbf{b}} = W(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n\right)$. At every time-step the committer is allowed to either move the machine head to the right or to restart its position to 0.

With the above notation, we can now view $\mathsf{MLE}(W, \boldsymbol{\zeta} \in \mathbb{F}^n)$ as an inner-product between W and $Z = (z_{\mathbf{b}} = \overline{\beta}(\mathbf{b}, \boldsymbol{\zeta}) : \mathbf{b} \in \{0,1\}^n)$ where computing $z_{\mathbf{b}}$ requires $O(n = \log N)$ field multiplications for fixed $\boldsymbol{\zeta}$ any $\mathbf{b} \in \{0,1\}^n$.

3.4 Polynomial Commitment Scheme to Multi-linear Extensions

Polynomial commitment schemes, introduced by Kate et al. [32] and generalized in [17,44,48], are a cryptographic primitive that allows one to commit to a multivariate polynomial of bounded degree and later provably reveal evaluations of the committed polynomial. Since we consider only multi-linear polynomials, we tailor our definition to them.

Convention. In defining the syntax of various protocols, we use the following convention for any list of arguments or returned tuple $(a, b, c; d, e)$ – variables listed before semicolon are known both to the prover and verifier whereas the ones after are only known to the prover. In this case, a, b, c are public whereas d, e are secret. In the absence of secret information the semicolon is omitted.

Definition 8 (Commitment to Multi-linear Extensions). *A polynomial commitment to multi-linear extensions is a tuple of protocols* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$*:*

1. $pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda, 1^N)$ *takes as input the unary representations of security parameter* $\lambda \in \mathbb{N}$ *and size parameter* $N = 2^n$ *corresponding to* $n \in \mathbb{N}$*, and produces public parameter* pp*. We allow* pp *to contain the description of the field* $\mathbb{F}$ *over which the multi-linear polynomials will be defined.*
2. $(\mathsf{C}; d) \xleftarrow{\$} \mathsf{Com}^H(pp, Y)$ *takes as input public parameter* pp *and sequence* $Y = (y_{\mathbf{b}} : \mathbf{b} \in \{0,1\}^n) \in \mathbb{F}^N$ *that defines the multi-linear polynomial to be committed, and outputs public commitment* C *and secret decommitment* d*.*
3. $b \leftarrow \mathsf{Open}^H(pp, \mathsf{C}, Y, d)$ *takes as input* pp*, a commitment* C*, sequence committed* Y *and a decommitment* d *and returns a decision bit* $b \in \{0,1\}$*.*
4. $\mathsf{Eval}^H(pp, \mathsf{C}, \boldsymbol{\zeta}, \gamma; Y, d)$ *is a public-coin interactive protocol between a prover* P *and a verifer* V *with common inputs—public parameter* pp*, commitment* C*, evaluation point* $\boldsymbol{\zeta} \in \mathbb{F}^n$ *and claimed evaluation* $\gamma \in \mathbb{F}$*, and prover has secret inputs* Y *and* d*. The prover then engages with the verifier in an interactive argument system for the relation*

$$\mathcal{R}_{\mathsf{mle}}(pp) = \left\{ (\mathsf{C}, \boldsymbol{\zeta}, \gamma; Y, d) : \mathsf{Open}^H(pp, \mathsf{C}, Y, d) = 1 \wedge \gamma = \mathsf{MLE}(Y, \boldsymbol{\zeta}) \right\}. \quad (6)$$

The output of V *is the output of* Eval *protocol.*

Furthermore, we require the following three properties.

1. *Computational Binding. For all PPT adversaries A and $n \in \mathbb{N}$*

$$\Pr\left[b_0 = b_1 \neq 0 \wedge Y_0 \neq Y_1 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda, 1^N) \\ (\mathsf{C}, Y_0, Y_1, d_0, d_1) \xleftarrow{\$} A^H(pp) \\ b_0 \leftarrow \mathsf{Open}^H(pp, \mathsf{C}, Y_0, d_0) \\ b_1 \leftarrow \mathsf{Open}^H(pp, \mathsf{C}, Y_1, d_1) \end{array}\right] \leq \mathsf{negl}(\lambda)\ .$$

2. *Perfect Correctness. For all $n, \lambda \in \mathbb{N}$ and all $Y \in \mathbb{F}^N$ and $\boldsymbol{\zeta} \in \mathbb{F}^n$,*

$$\Pr\left[1 = \mathsf{Eval}^H(pp, \mathsf{C}, Z, \gamma; Y, d) : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda, 1^N), \\ (\mathsf{C}; d) \xleftarrow{\$} \mathsf{Com}^H(pp, Y),\ \gamma = \mathsf{MLE}(Y, \boldsymbol{\zeta}) \end{array}\right] = 1\ .$$

3. *Witness-extended Emulation. We say that the polynomial commitment scheme has witness-extended emulation if* Eval *has a witness-extended emulation as an interactive argument for the relation ensemble $\{\mathcal{R}_{\mathsf{mle}}(pp)\}_{pp}$ (Eq. (6)) except with negligible probability over the choice of H and coins of $pp \xleftarrow{\$} \mathsf{Setup}^H(1^\lambda, 1^N)$.*

4 Space-Efficient Commitment for Multi-linear Extensions

In this section we describe our polynomial commitment scheme for multilinear extensions, a high level overview of which was provided in Sect. 2.1. We dedicate the remainder of the section to proving our main theorem:

Theorem 5. *Let* GGen *be a generator of obliviously sampleable, prime-order groups. Assuming the hardness of discrete logarithm problem for* GGen*, the scheme* (Setup, Com, Open, Eval) *defined in Sect. 4.1 is a polynomial commitment scheme to multi-linear extensions with witness-extended emulation in the random oracle model. Furthermore, for every $N \in \mathbb{N}$ and sequence $Y \in \mathbb{F}^N$, the committer/prover has multi-pass streaming access to Y and*

1. Com *performs $O(N \log p)$ group operations, stores $O(1)$ field and group elements, requires one pass over Y, makes N queries to the random oracle, and outputs a single group element. Evaluating $\mathsf{MLE}(Y, \cdot)$ requires $O(N)$ field operations, storing $O(1)$ field elements and requires one pass over Y.*
2. Eval *is public-coin and has $O(\log N)$ rounds with $O(1)$ group elements sent in every round. Furthermore,*
   - *Prover performs $O(N \cdot (\log^2 N) \cdot \log p)$ field and group operations, $O(N \log N)$ queries to the random oracle, requires $O(\log N)$ passes over Y and stores $O(\log N)$ field and group elements.*
   - *Verifier performs $O(N \cdot (\log N) \cdot \log p)$ field and group operations, $O(N)$ queries to the random oracle, and stores $O(\log N)$ field and group elements.*

Section 4.1 describes our scheme, Sect. 4.2 and Sect. 4.3 establish its security and efficiency.

$\mathsf{Eval}(pp, \mathsf{C}, \boldsymbol{\zeta}, \gamma; Y)$

1 : V samples and sends $\mathsf{g} \xleftarrow{\$} \mathbb{G}$

2 : P and V define $\mathsf{C}_\gamma \leftarrow \mathsf{C} \cdot \mathsf{g}^\gamma$

3 : P and V define the sequence $Z = \left(z_{\mathbf{b}} = \prod_{i=1}^{n} \beta(b_i, \zeta_i) : \mathbf{b} \in \{0,1\}^{n-1}\right)$

4 : P and V engage in $\mathsf{EvalReduce}(\mathsf{C}_\gamma, Z, \gamma, \mathbf{g}, \mathsf{g}; Y)$

$\mathsf{EvalReduce}(\mathsf{C}_\gamma \in \mathbb{G}, Z = (z_{\mathbf{b}}), \gamma \in \mathbb{F}, \mathbf{g} = (\mathsf{g}_{\mathbf{b}}), \mathsf{g}; Y = (y_{\mathbf{b}}))$

proves knowledge of Y **such that:** $\mathsf{C}_\gamma = \mathsf{Com}(\mathbf{g}, Y) \cdot \mathsf{g}^\gamma$ **and** $\langle Y, Z \rangle = \gamma$**.**

1 : $N \leftarrow |Z|, n \leftarrow \log N$

2 : **if** $N = 1$: **then**

3 : Let $\mathbf{g} = (\mathsf{g}'),\ Z = (z),\ Y = (y)$

4 : P sends y to V who accepts iff $\mathsf{C}_\gamma = \mathsf{g}'^{y} \cdot \mathsf{g}^{y \cdot z}$

5 : **else**

6 : P computes γ_{L} and γ_{R} where

$$\gamma_{\mathsf{L}} \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{n-1}} y_{\mathbf{b}0} \cdot z_{\mathbf{b}1}\ ;\ \gamma_{\mathsf{R}} \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{n-1}} y_{\mathbf{b}1} \cdot z_{\mathbf{b}0}.$$

7 : P computes and sends C_{L} and C_{R} where

$$\mathsf{C}_{\mathsf{L}} \leftarrow \mathsf{g}^{\gamma_{\mathsf{L}}} \cdot \prod_{\mathbf{b} \in \{0,1\}^{n-1}} (\mathsf{g}_{\mathbf{b}1})^{y_{\mathbf{b}0}}\ ;\ \mathsf{C}_{\mathsf{R}} \leftarrow \mathsf{g}^{\gamma_{\mathsf{R}}} \cdot \prod_{\mathbf{b} \in \{0,1\}^{n-1}} (\mathsf{g}_{\mathbf{b}0})^{y_{\mathbf{b}1}}.$$

8 : V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends it to P.

9 : P computes and sends $\gamma' = \alpha^2 \cdot \gamma_{\mathsf{L}} + \gamma + \alpha^{-2} \cdot \gamma_{\mathsf{R}}$.

10 : P and V both compute

$$\mathsf{C}'_{\gamma'} \leftarrow (\mathsf{C}_{\mathsf{L}})^{\alpha^2} \cdot \mathsf{C}_\gamma \cdot (\mathsf{C}_{\mathsf{R}})^{\alpha^{-2}},$$
$$Z' = \left(z'_{\mathbf{b}} = \alpha^{-1} \cdot z_{\mathbf{b}0} + \alpha \cdot z_{\mathbf{b}1}\right)_{\mathbf{b} \in \{0,1\}^{n-1}},$$
$$\mathbf{g}' = \left(\mathsf{g}'_{\mathbf{b}} = (\mathsf{g}_{\mathbf{b}0})^{\alpha^{-1}} \cdot (\mathsf{g}_{\mathbf{b}1})^{\alpha}\right)_{\mathbf{b} \in \{0,1\}^{n-1}}.$$

11 : P computes $Y' = \left(y'_{\mathbf{b}} = \alpha \cdot y_{\mathbf{b}0} + \alpha^{-1} \cdot y_{\mathbf{b}1}\right)_{\mathbf{b} \in \{0,1\}^{n-1}}$.

12 : **return** $\mathsf{EvalReduce}(\mathsf{C}'_{\gamma'}, Z', \gamma', \mathbf{g}', \mathsf{g}; Y')$

Fig. 3. Eval protocol for the commitment scheme from Sect. 4.1.

4.1 Commitment Scheme

We describe a commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ to multi-linear extensions below.

1. $\mathsf{Setup}^H(1^\lambda, 1^N)$: On inputs security parameter 1^λ and size parameter $N = 2^n$ and access to H, Setup samples $(\mathbb{G}, p, \mathsf{g}) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, sets $\mathbb{F} = \mathbb{F}_p$ and returns

$pp = (\mathbb{G}, \mathbb{F}, N, p)$. Furthermore, it implicitly defines a sequence of generators $\mathbf{g} = (\mathsf{g}_{\mathbf{b}} = H(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n)$.

2. $\mathsf{Com}^H(pp, Y)$ returns $\mathsf{C} \in \mathbb{G}$ as the commitment and Y as the decommitment where
$$\mathsf{C} \leftarrow \prod_{\mathbf{b} \in \{0,1\}^n} (\mathsf{g}_{\mathbf{b}})^{y_{\mathbf{b}}} .$$
3. $\mathsf{Open}^H(pp, \mathsf{C}, Y)$ returns 1 iff $\mathsf{C} = \mathsf{Com}^H(pp, Y)$.
4. $\mathsf{Eval}^H(pp, \mathsf{C}, \boldsymbol{\zeta}, \gamma; Y)$ is an interactive protocol $\langle P, V \rangle$ that begins with V sending a random $\mathsf{g} \xleftarrow{\$} \mathbb{G}$. Then, both P and V compute the commitment $\mathsf{C}_\gamma \leftarrow \mathsf{C} \cdot \mathsf{g}^\gamma$ to additionally bind the claimed evaluation γ. Then, P and V engage in an interactive protocol $\mathsf{EvalReduce}$ on input $(\mathsf{C}_\gamma, Z, \mathbf{g}, \mathsf{g}, \gamma; Y)$ where the prover proves knowledge of Y such that
$$\mathsf{C}_\gamma = \mathsf{Com}(\mathbf{g}, Y) \cdot \mathsf{g}^\gamma \wedge \langle Y, Z \rangle = \gamma,$$
where $Z = (z_{\mathbf{b}} = \bar{\beta}(\mathbf{b}, \boldsymbol{\zeta}) : \mathbf{b} \in \{0,1\}^n)$. We define the protocol in Fig. 3.

Remark 3. In fact, our scheme readily extends to proving any linear relation $\boldsymbol{\alpha}$ about a committed sequence Y (i.e., the value $\langle \boldsymbol{\alpha}, Y \rangle$), as long as each element of $\boldsymbol{\alpha}$ can be generated in poly-logarithmic time.

4.2 Correctness and Security

Lemma 2. *The scheme from Sect. 4.1 is perfectly correct, computationally binding and* Eval *has witness-extended emulation under the hardness of the discrete logarithm problem for groups sampled by* GGen *in the random oracle model.*

The perfect correctness of the scheme follows from the correctness of $\mathsf{EvalReduce}$ protocol, which we prove in Lemma 3, computationally binding follows from that of Pedersen multi-commitments which follows from the hardness of discrete-log (in the random oracle model). The witness-extended emulation of Eval follows from the witness-extended emulation of the inner-product protocol in [15]. At a high level, we make two changes to their inner-product protocol: (1) sample the generators using the random oracle H, (2) perform the 2-move reduction step using the lsb-based folding approach (see Sect. 2.1 for a discussion). At a high level, given a witness Y for the inner-product statement $(\mathsf{C}_\gamma, \mathbf{g}, Z, \gamma)$, one can compute a witness for the permuted statement $(\mathsf{C}_\gamma, \pi(\mathbf{g}), \pi(Z), \gamma)$ for any efficiently computable/invertible public permutation π. Choosing π as the permutation that reverses its input allows us, in principle, to base the extractability of our scheme (lsb-based folding) to the original scheme of [15]. We provide a formal proof in the full version. Due to (1) our scheme enjoys security only in the random-oracle model.

Lemma 3. *Let* $(\mathsf{C}_\gamma, Z, \gamma, \mathbf{g}, \mathsf{g}; Y)$ *be inputs to* $\mathsf{EvalReduce}$ *and let* $(\mathsf{C}'_{\gamma'}, Z', \gamma', \mathbf{g}', \mathsf{g}; Y')$ *be generated as in Fig. 3. Then,*
$$\begin{array}{c} \mathsf{C}_\gamma = \mathsf{Com}(\mathbf{g}, Y) \cdot \mathsf{g}^\gamma \\ \wedge \\ \langle Y, Z \rangle = \gamma \end{array} \implies \begin{array}{c} \mathsf{C}'_{\gamma'} = \mathsf{Com}(\mathbf{g}', Y') \cdot \mathsf{g}^{\gamma'} \\ \wedge \\ \langle Y', Z' \rangle = \gamma' \end{array} .$$

Proof. Let $N = |Z|$ and let $n = \log N$. Then,

1. To show $\gamma' = \langle Y', Z' \rangle$:

$$\begin{aligned}
\langle Y', Z' \rangle &= \sum_{\mathbf{b} \in \{0,1\}^{n-1}} y'_{\mathbf{b}} \cdot z'_{\mathbf{b}}, \\
&= \sum_{\mathbf{b} \in \{0,1\}^{n-1}} (\alpha \cdot y_{\mathbf{b}0} + \alpha^{-1} \cdot y_{\mathbf{b}1}) \cdot (\alpha^{-1} \cdot z_{\mathbf{b}0} + \alpha \cdot z_{\mathbf{b}1}), \\
&= \sum_{\mathbf{b} \in \{0,1\}^{n-1}} y_{\mathbf{b}0} \cdot z_{\mathbf{b}0} + \alpha^2 \cdot y_{\mathbf{b}0} \cdot z_{\mathbf{b}1} + y_{\mathbf{b}1} \cdot z_{\mathbf{b}1} + \alpha^{-2} \cdot y_{\mathbf{b}1} \cdot z_{\mathbf{b}1}, \\
&= \gamma + \alpha^2 \cdot \gamma_{\mathsf{L}} + \alpha^{-2} \cdot \gamma_{\mathsf{R}} = \gamma'.
\end{aligned}$$

2. $\mathsf{C}'_{\gamma'} = \mathsf{Com}(\mathbf{g}', Y') \cdot \mathsf{g}^{\gamma'}$:

$$\begin{aligned}
\mathsf{Com}(\mathbf{g}', Y') &= \prod_{\mathbf{b} \in \{0,1\}^{n-1}} (\mathsf{g}'_{\mathbf{b}})^{y'_{\mathbf{b}}}, = \prod_{\mathbf{b} \in \{0,1\}^{n-1}} \left(\mathsf{g}_{\mathbf{b}0}^{\alpha^{-1}} \cdot g_{\mathbf{b}1}^{\alpha} \right)^{\alpha \cdot y_{\mathbf{b}0} + \alpha^{-1} \cdot y_{\mathbf{b}1}}, \\
&= \prod_{\mathbf{b} \in \{0,1\}^{n-1}} \left(\mathsf{g}_{\mathbf{b}0}^{y_{\mathbf{b}0}} \cdot g_{\mathbf{b}0}^{\alpha^{-2} \cdot y_{\mathbf{b}1}} \cdot \mathsf{g}_{\mathbf{b}1}^{\alpha^2 \cdot y_{\mathbf{b}0}} \cdot \mathsf{g}_{\mathbf{b}1}^{y_{\mathbf{b}1}} \right), \\
&= \prod_{\mathbf{b} \in \{0,1\}^{n-1}} (\mathsf{g}_{\mathbf{b}0}^{y_{\mathbf{b}1}})^{\alpha^{-2}} \cdot \mathsf{g}_{\mathbf{b}0}^{y_{\mathbf{b}0}} \cdot \mathsf{g}_{\mathbf{b}1}^{y_{\mathbf{b}1}} \cdot (\mathsf{g}_{\mathbf{b}1}^{y_{\mathbf{b}0}})^{\alpha^2}.
\end{aligned}$$

Then, above with the definition of γ' implies that $\mathsf{C}'_{\gamma'} = \mathsf{Com}(\mathbf{g}', Y') \cdot \mathsf{g}^{\gamma'}$.

4.3 Efficiency

In this section we discuss the efficiency aspects of each of the protocols defined in Sect. 4.1 with respect to four complexity measures: (1) queries to the random oracle H, (2) field/group operations performed, (3) field/group elements stored and (4) number of passes over the stream Y.

For the rest of this section, we fix $n, N = 2^n, H, \mathbb{G}, \mathbb{F}, \boldsymbol{\zeta} \in \mathbb{F}^n$ and furthermore fix $Y = (y_{\mathbf{b}} : \mathbf{b} \in \{0,1\}^n)$, $\mathsf{g} = (\mathsf{g}_{\mathbf{b}} = H(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n)$ and $Z = (z_{\mathbf{b}} = \bar{\beta}(\mathbf{b}, \boldsymbol{\zeta}) : \mathbf{b} \in \{0,1\}^n)$. Note given $\boldsymbol{\zeta}$, any $z_{\mathbf{b}}$ can be computed by performing $O(n)$ field operations.

First, consider the prover P of Eval protocol (Fig. 3). Given the inputs $(\mathsf{C}, Z, \gamma, \mathbf{g}, \mathsf{g}; Y)$, P and V call the recursive protocol $\mathsf{EvalReduce}$ on the N sized statement $(\mathsf{C}_\gamma, Z, \gamma, \mathbf{g}, \mathsf{g}; Y)$ where $\mathsf{C}_\gamma = \mathsf{C} \cdot \mathsf{g}^\gamma$. The prover's computation in this call to $\mathsf{EvalReduce}$ is dictated by computing (a) $\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}}$ (line 6), (2) $\mathsf{C}_{\mathsf{L}}, \mathsf{C}_{\mathsf{R}}$ (line 7) and (c) inputs for the next recursive call on $\mathsf{EvalReduce}$ with $N/2$ sized statement $(\mathsf{C}'_{\gamma'}, Z', \gamma', \mathbf{g}', \mathsf{g}; Y')$ (line 9,11). The rest of its computation requires $O(1)$ number of operations. The recursion ends on the n-th call with statement of size 1. For $k \in \{0, \ldots, n\}$, the inputs at the k-th depth of the recursion be denoted with superscript k, that is, $\mathsf{C}^{(k)}, \gamma^{(k)}, Z^{(k)}, \mathsf{g}^{(k)}, Y^{(k)}$. For example,

$\mathsf{Computez}(k, \mathbf{c}, \boldsymbol{\zeta}, \boldsymbol{\alpha})$

1 : $z_{\mathbf{c}}^{(k)} \leftarrow 0$
2 : **foreach** $\mathbf{a} \in \{0,1\}^k$ **do**
3 : $\quad \mathsf{temp} \leftarrow 1_{\mathbb{F}}$
4 : $\quad$ **foreach** $j \in \{1, \ldots, k\}$ **do**
5 : $\quad\quad \mathsf{temp} \leftarrow \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j-1)}, a_j)$
6 : $\quad z_{\mathbf{c}}^{(k)} \leftarrow \mathsf{temp} \cdot \beta(\mathbf{c} \circ \mathbf{a}, \boldsymbol{\zeta})$
7 : **return** $z_{\mathbf{c}}^{(k)}$

$\mathsf{Computeg}^H(k, \mathbf{c}, \boldsymbol{\alpha})$

1 : $\mathbf{g}_{\mathbf{c}}^{(k)} \leftarrow 0$
2 : **foreach** $\mathbf{a} \in \{0,1\}^k$ **do**
3 : $\quad \mathsf{temp} \leftarrow 1_{\mathbb{F}}$
4 : $\quad$ **foreach** $j \in \{1, \ldots, k\}$ **do**
5 : $\quad\quad \mathsf{temp} \leftarrow \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j-1)}, a_j)$
6 : $\quad \mathbf{g}_{\mathbf{c}}^{(k)} \leftarrow H(\mathbf{c} \circ \mathbf{a})^{\mathsf{temp}}$
7 : **return** $\mathbf{g}_{\mathbf{c}}^{(k)}$

Fig. 4. Algorithms for computing $z_{\mathbf{b}}^{(k)}$ and $\mathbf{g}_{\mathbf{b}}^{(k)}$. In both algorithms $\mathbf{c} \in \{0,1\}^{n-k}$ and $\boldsymbol{\alpha} = (\alpha^{(0)}, \ldots, \alpha^{(k-1)})$, where $\beta(\mathbf{b}, \boldsymbol{\zeta}) = \prod_{i=1}^{n} \beta(b_i, \zeta_i)$ for $\mathbf{b} = \mathbf{c} \circ \mathbf{a}$ and $\mathsf{coeff}(\alpha, c) = \alpha \cdot c + \alpha^{-1} \cdot (1 - c)$.

$Z^{(0)} = Z$, $Y^{(0)} = Y$ denote the initial inputs (at depth 0) where prover computes $\gamma_{\mathsf{L}}^{(0)}, \gamma_{\mathsf{R}}^{(0)}, \mathsf{C}_{\mathsf{L}}^{(0)}, \mathsf{C}_{\mathsf{R}}^{(0)}$ with verifier challenge $\alpha^{(0)}$. The sequences $Z^{(k)}, Y^{(k)}$ and $\mathbf{g}^{(k)}$ are of size 2^{n-k}.

At a high level, we ask prover to never explicitly compute the sequences $\mathbf{g}^{(k)}, Z^{(k)}, Y^{(k)}$ (item (c) above) but instead compute elements $\mathbf{g}_{\mathbf{b}}^{(k)}, z_{\mathbf{b}}^{(k)}, y_{\mathbf{b}}^{(k)}$, of the respective sequences, on demand, which then can be used to compute $\gamma_{\mathsf{L}}^{(k)}, \gamma_{\mathsf{R}}^{(k)}, \mathsf{C}_{\mathsf{L}}^{(k)}, \mathsf{C}_{\mathsf{R}}^{(k)}$ in required time and space. For this, first it will be useful to see how the elements of sequences $Z^{(k)}, Y^{(k)}, \mathbf{g}^{(k)}$ depend on the initial (i.e., depth-0) sequence $Z^{(0)}, Y^{(0)}, \mathbf{g}^{(0)}$.

Relating $Y^{(k)}$ with $Y^{(0)}$. First, lets consider $Y^{(k)} = (y_{\mathbf{b}}^{(k)} : \mathbf{b} \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$. Let $(\alpha^{(0)}, \ldots, \alpha^{(k-1)})$ be the verifier's challenges sent in all prior rounds.

Lemma 4 (Streaming of $Y^{(k)}$). *For every* $\mathbf{b} \in \{0,1\}^{n-k}$,

$$y_{\mathbf{b}}^{(k)} = \sum_{\mathbf{c} \in \{0,1\}^k} \left(\prod_{j=1}^{k} \mathsf{coeff}(\alpha^{(j-1)}, c_j) \right) \cdot y_{\mathbf{b} \circ \mathbf{c}}, \tag{7}$$

where $\mathsf{coeff}(\alpha, c) = \alpha \cdot (1 - c) + \alpha^{-1} \cdot c$.

The proof follows by induction on depth k. Lemma 4 allows us to simulate the stream $Y^{(k)}$ with **one** pass over the initial sequence Y, additionally performing $O(N \cdot k)$ multiplications to compute appropriate coeff functions.

Relating $Z^{(k)}$ *with* $Z^{(0)}$. Next, consider $Z^{(k)} = (z_{\mathbf{b}}^{(k)} : \mathbf{b} \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$.

Lemma 5 (Computing $z_{\mathbf{b}}^{(k)}$). *For every* $\mathbf{b} \in \{0,1\}^{n-k}$,

$$z_{\mathbf{b}}^{(k)} = \sum_{\mathbf{c}\in\{0,1\}^k} \left(\prod_{j=1}^{k} \mathsf{coeff}(\alpha^{(j-1)}, c_j) \right) \cdot z_{\mathbf{b}\circ\mathbf{c}}, \tag{8}$$

where $\mathsf{coeff}(\alpha, c) = \alpha \cdot c + \alpha^{-1} \cdot (1-c)$. *Furthermore, computing* $z_{\mathbf{b}}^{(k)}$ *requires* $O(2^k \cdot n)$ *field multiplications and storing* $O(n)$ *elements (see algorithm* Computez *in Fig. 4).*

Relating $\mathrm{g}^{(k)}$ *with* $\mathrm{g}^{(0)}$. Finally, consider $\mathbf{g}^{(k)} = (\mathsf{g}_{\mathbf{b}}^{(k)} : \mathbf{b} \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$.

Lemma 6 (Computing $\mathsf{g}_{\mathbf{b}}^{(k)}$). *For every* $\mathbf{b} \in \{0,1\}^{n-k}$,

$$\mathsf{g}_{\mathbf{b}}^{(k)} = \prod_{\mathbf{c}\in\{0,1\}^k} \mathsf{g}_{\mathbf{b}\circ\mathbf{c}}^{\mathsf{coeff}(\boldsymbol{\alpha},\mathbf{c})} \;;\; \mathsf{coeff}(\boldsymbol{\alpha}, \mathbf{c}) = \prod_{i=1}^{k} \alpha^{(j-1)} \cdot c_j + (\alpha^{(j-1)})^{-1} \cdot (1-c_j). \tag{9}$$

Furthermore, computing $\mathsf{g}_{\mathbf{b}}^{(k)}$ *requires* $2^k \cdot k$ *field multiplications,* 2^k *queries to* H, 2^k *group multiplications and exponentiations, and storing* $O(k)$ *elements (see algorithm* Computeg *in Fig. 4).*

We now discuss the efficiency of the commitment scheme.

Commitment Phase. We first note that Com^H on input pp and given streaming access to Y can compute the commitment $\mathsf{C} = \prod_{\mathbf{b}} (H(\mathbf{b}))^{y_{\mathbf{b}}}$ for $\mathbf{b} \in \{0,1\}^n$ making N queries to H, performing N group exponentiations and a single pass over Y. Furthermore, requires storing only a single group element.

Note that a single group exponentiation g^{α} can be emulated while performing $O(\log p)$ group multiplications while storing $O(1)$ group and field elements. Since, $\mathbb{G}, \mathbb{F}$ are of order p, field and group operations can, furthermore, be performed in polylog$(p(\lambda))$ time.

Evaluating $\mathsf{MLE}(\boldsymbol{Y}, \boldsymbol{\zeta})$. The honest prover (when used in higher level protocols) needs to evaluate $\mathsf{MLE}(Y, \boldsymbol{\zeta})$ which requires performing $O(N \log N)$ field operations overall and a single pass over stream Y.

Prover Efficiency. For every depth-k of the recursion, it is sufficient to discuss the efficiency of computing $\gamma_{\mathsf{L}}^{(k)}, \gamma_{\mathsf{R}}^{(k)}, \mathsf{C}_{\mathsf{L}}^{(k)}, \mathsf{C}_{\mathsf{R}}^{(k)}$. We argue the complexity of computing $\gamma_{\mathsf{L}}^{(k)}$ and $\mathsf{C}_{\mathsf{L}}^{(k)}$ and the analysis for the remaining is similar. We give a formal algorithm Prover in Fig. 5.

Computing $\gamma_{\mathsf{L}}^{(k)}$. Recall that $\gamma_{\mathsf{L}}^{(k)} = \sum_{\mathbf{b}} y_{\mathbf{b}0}^{(k)} \cdot z_{\mathbf{b}1}^{(k)}$ for $\mathbf{b} \in \{0,1\}^{n-k-1}$. To compute $\gamma_{\mathsf{L}}^{(k)}$ we stream the initial N-sized sequence Y and generate elements of

$\mathsf{Prover}^H(pp, k, Y, \boldsymbol{\zeta}, \mathsf{g}, \alpha^{(0)}, \ldots, \alpha^{(k-1)})$

1 : $\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}}, y^{(k)} \leftarrow 0_{\mathbb{F}}, \mathsf{g}^{(k)}, \mathsf{C}_{\mathsf{L}}, \mathsf{C}_{\mathsf{R}} \leftarrow 1_{\mathbb{G}}, count \leftarrow 0$
2 : **foreach** $\mathbf{b} = (b_n, \ldots, b_1) \in \{0,1\}^n$ **do**
3 : $\quad \mathsf{temp} \leftarrow 1_{\mathbb{F}}$
4 : $\quad$ **foreach** $j \in \{1, \ldots, k\}$ **do**
5 : $\quad\quad \mathsf{temp} \leftarrow \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j-1)}, b_j)$
6 : $\quad y^{(k)} \leftarrow y^{(k)} + \mathsf{temp} \cdot y_{\mathbf{b}}$
7 : $\quad count \leftarrow count + 1$
8 : $\quad$ **if** $count == 2^k$ **then**
9 : $\quad\quad z^{(k)} \leftarrow \mathsf{Computez}(k, (b_n, \ldots, b_{n-k+1}, 1 - b_{n-k}), \boldsymbol{\zeta}, \alpha^{(0)}, \ldots, \alpha^{(k-1)})$
10 : $\quad\quad \mathsf{g}^{(k)} \leftarrow \mathsf{Computeg}^H(k, (b_n, \ldots, b_{n-k+1}, 1 - b_{n-k}), \alpha^{(0)}, \ldots, \alpha^{(k-1)})$
11 : $\quad\quad$ **if** $b_{n-k} == 0$ **then**
12 : $\quad\quad\quad \gamma_{\mathsf{L}} \leftarrow \gamma_{\mathsf{L}} + z^{(k)} \cdot y^{(k)}$; $\mathsf{C}_{\mathsf{L}} \leftarrow \mathsf{C}_{\mathsf{L}} \cdot (\mathsf{g}^{(k)})^{y^{(k)}}$
13 : $\quad\quad$ **else**
14 : $\quad\quad\quad \gamma_{\mathsf{R}} \leftarrow \gamma_{\mathsf{R}} + z^{(k)} \cdot y^{(k)}$; $\mathsf{C}_{\mathsf{R}} \leftarrow \mathsf{C}_{\mathsf{R}} \cdot (\mathsf{g}^{(k)})^{y^{(k)}}$
15 : $\quad\quad y^{(k)} \leftarrow 0_{\mathbb{F}}$; $\mathsf{g}^{(k)} \leftarrow 1_{\mathbb{G}}$; $count \leftarrow 0$
16 : $\mathsf{C}_{\mathsf{L}} \leftarrow \mathsf{C}_{\mathsf{L}} \cdot \mathsf{g}^{\gamma_{\mathsf{L}}}$; $\mathsf{C}_{\mathsf{R}} \leftarrow \mathsf{C}_{\mathsf{R}} \cdot \mathsf{g}^{\gamma_{\mathsf{R}}}$
17 : **return** $(\gamma_{\mathsf{L}}, \mathsf{C}_{\mathsf{L}}, \gamma_{\mathsf{R}}, \mathsf{C}_{\mathsf{R}})$

Fig. 5. Space-efficient prover

the sequence $(y^{(k)}_{\mathbf{b}0} : \mathbf{b} \in \{0,1\}^{n-k-1})$ in a streaming manner. Since each $y^{(k)}_{\mathbf{b}0}$ depends on a contiguous block of 2^k elements in the initial stream Y, we can compute $y^{(k)}_{\mathbf{b}0}$ by performing $2^k \cdot k$ field operations (lines 2–7 in Fig. 5). For every $\mathbf{b} \in \{0,1\}^{n-k-1}$, after computing $y^{(k)}_{\mathbf{b}0}$, we leverage "random access" to Z and compute $z^{(k)}_{\mathbf{b}1}$ (Lemma 5) which requires $O(2^k \cdot k)$ field operations. Overall, $\gamma^{(k)}_{\mathsf{L}}$ can be computed in $O(N \cdot k)$ field operations and a single pass over Y.

Computing $\mathsf{C}^{(k)}_{\mathsf{L}}$. The two differences in computing $\mathsf{C}^{(k)}_{\mathsf{L}}$ (see Fig. 3 for the definition) is that (a) we need to compute $\mathsf{g}^{(k)}_{\mathbf{b}1}$ instead of computing $z^{(k)}_{\mathbf{b}1}$ and (b) perform group exponentiations, that is, $\mathsf{g}^{(k)y^{(k)}_{\mathbf{b}0}}_{\mathbf{b}1}$ as opposed to group multiplications as in the computation of $\gamma^{(k)}_{\mathsf{L}}$. Both steps overall can be implemented in $O(N \cdot k \cdot \log p)$ field and group operations and N queries to H (Lemma 6). Overall, at depth k the prover (1) makes $O(N)$ queries to H, (2) performs $O(N \cdot k \cdot \log(p))$ field and group operations and (3) requires a single pass over Y.

Therefore, the entire prover computation (over all calls to $\mathsf{EvalReduce}$) requires $O(\log N)$ passes over Y, makes $O(N \log N)$ queries to H and performs $O(N \cdot \log^2 N \cdot \log p)$ field/group operations. Furthermore, this requires storing only $O(\log N)$ field and group elements.

Verifier Efficiency. V only needs to compute folded sequence $Z^{(n)}$ and folded generators $\mathsf{g}^{(n)}$ at depth-n of the recursion. These can computed by invoking Computez and Computeg (Fig. 4) with $k = n$ and require $O(N \cdot \log(N, p))$ field and group operations, $O(N)$ queries to H and storing $O(\log N)$ field and group elements.

Lemma 7. *The time and space efficiency of each of the phases of the protocols are listed below*[6]*:*

Computation	H queries	Y passes	$\mathbb{F}/\mathbb{G}$ ops	$\mathbb{G}/\mathbb{F}$ elements
Com	N	1	$O(N)$	O(1)
MLE(Y, ζ)	0	1	$O(N \log N)$	O(1)
P (in Eval)	$O(N \log N)$	$O(\log N)$	$O(N \log^2 N)$	$O(\log N)$
V (in Eval)	$O(N)$	0	$O(N \log N)$	$O(\log N)$

Finally, Theorem 5 follows directly from Lemma 2 and Lemma 7.

5 A Polynomial IOP for Random Access Machines

We obtain space efficient arguments for any NP relation verifiable by time-T space-S RAM computations by compiling our polynomial commitment scheme with a suitable space-efficient *polynomial interactive oracle proof* (IOP) [6,17,41]. Informally, a polynomial IOP is a multi-round interactive PCP such that in each round the verifier sends a message to the prover and the prover responds with a proof oracle that the verifier can query via random access, with the additional property that the proof oracle is a polynomial.

We dedicate the remainder of this section to giving a high-level overview our polynomial IOP (PIOP), presented in Fig. 6, which realizes Theorem 3. Full details are deferred to the full-version. We first recall that we consider a variant of the polynomial IOP model in which all prover messages are encoded by a channel and that the prover does not incur the cost of this encoding in its time and space complexity. In particular, we consider a channel which computes the multi-linear extension of the prover messages. Our space-efficient PIOP leverages the RAM to circuit satisfiability reduction of [12]: this RAM to circuit reduction outputs an arithmetic circuit of size $T \cdot \text{polylog}(T)$, which we denote as C_M, over finite field $\mathbb{F}$ of size $\text{polylog}(T)$. The circuit is defined such that such that $C_M(x) = y$ if and only if $M(x; w) = y$ for auxiliary input w. Further, the circuit has a "streaming" property: the string of gate assignments W of C_M on input x can be computed "gate-by-gate" in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$. In our model, this allows our prover to stream its message through the encoding channel in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$ and send the verifier

[6] $\log(p)$ factors are omitted.

$\mathsf{PIOP}(M, x, T, S; w)$

1 : P compiles circuit C_M and transcript W via the reduction of [12].

2 : P provides V with an oracle for $\overline{W}$.

3 : V samples $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$ and sends $\boldsymbol{\tau}$ to P.

4 : P computes polynomial $h_{\boldsymbol{\tau}}$ and sets $\gamma \leftarrow 0$. P sends γ to V.

5 : V sets $\gamma' \leftarrow \gamma$.

6 : **foreach** $j \in \{1, \ldots, 3s\}$ **do** // sum-check

7 : P sends $h_{\boldsymbol{\tau}}^{(j)}(X_j)$ to V, where $h_{\boldsymbol{\tau}}^{(j)}(X_j) \leftarrow \sum_{\mathbf{c}' \in \{0,1\}^{3s-j}} h_{\boldsymbol{\tau}}(\alpha_1, \ldots, \alpha_{j-1}, X_j, \mathbf{c}')$.

8 : V checks $\gamma' \stackrel{?}{=} h_{\boldsymbol{\tau}}^{(j)}(0) + h_{\boldsymbol{\tau}}^{(j)}(1)$, rejecting if equality doesn't hold.

9 : V samples $\alpha_j \xleftarrow{\$} \mathbb{F}$ and sets $\gamma' \leftarrow h_{\boldsymbol{\tau}}^{(j)}(\alpha_j)$.

10 : **if** $j < 3s$ **then** V sends α_j to P **endif**

11 : V queries the oracle $\overline{W}$ and obtains $\gamma_i \leftarrow \overline{W}(\boldsymbol{\alpha}^{(i)})$ for $i \in \{1, 2, 3\}$, where $\boldsymbol{\alpha}^{(i)} \leftarrow (\alpha_{i \cdot 1}, \ldots, \alpha_{i \cdot s})$.

12 : V computes $h_{\boldsymbol{\tau}}(\boldsymbol{\alpha})$ using oracle queries γ_i and accepts if and only if $\gamma' = h_{\boldsymbol{\tau}}(\boldsymbol{\alpha})$.

Fig. 6. Our Polynomial IOP for time-T space-S RAM computations.

with an oracle to the *multi-linear extension* of W, denoted as $\overline{W}$. We emphasize that $\overline{W}$ is the only oracle sent by the prover to the verifier, and that this and the streaming property of W are key to the composition of our PIOP with the polynomial commitment scheme of Theorem 5.

The circuit satisfiability instance (C_M, x, y) is next reduced to an algebraic claim about a constant-degree polynomial $F_{x,y}$ whose structure depends on the wiring pattern of C_M, x and y, and the oracle $\overline{W}$. The polynomial $F_{x,y}$ has the property that it is the 0-polynomial if and only if $\overline{W}$ is a multi-linear extension of a correct transcript; i.e., that W is a witness for $C_M(x) = y$. A verifier is convinced that $F_{x,y}$ is the 0-polynomial if $F_{x,y}(\boldsymbol{\tau}) = 0$ for uniformly random $\mathbb{F}$-vector $\boldsymbol{\tau}$. $F_{x,y}$ is suitably structured such that a prover can convince a verifier that $F_{x,y}(\boldsymbol{\tau}) = 0$ via the classical sum-check protocol [36,45]. In particular, the value $F_{x,y}(\boldsymbol{\tau})$ is expressed as a summation of some constant-degree polynomial $h_{\boldsymbol{\tau}}$ over the Boolean hypercube:

$$F_{x,y}(\boldsymbol{\tau}) = \sum_{\mathbf{c} \in \{0,1\}^n} h_{\boldsymbol{\tau}}(\mathbf{c}) \ .$$

The polynomial $h_{\boldsymbol{\tau}}$ has the following two key efficiency properties: (1) the prover's messages in the sum-check that depend on $h_{\boldsymbol{\tau}}$ are computable in $T \cdot \text{polylog}(T)$ time and space $S \cdot \text{polylog}(T)$ (see [12, Lemma 4.2], full details deferred to the full-version); and (2) given oracle $\overline{W}$ the verifier in time $\text{polylog}(T)$ can evaluate $h_{\boldsymbol{\tau}}$ at any point without explicit access to the circuit C_M (see [12, Theorem 4.1 and Lemma 4.2], full details deferred to the full-version).

6 Time- and Space-Efficient Arguments for RAM

We obtain space-efficient arguments $\langle P_{\mathsf{arg}}, V_{\mathsf{arg}} \rangle$ for NP relations that can be verified by time-T space-S RAMs by composing the polynomial commitment scheme of Theorem 5 and the polynomial IOP of Fig. 6. Specifically, the prover P_{arg} and V_{arg} runs the prover and the verifier of the underlying PIOP except two changes: (1) P_{arg} (line 2, Fig. 6) instead provides V_{arg} with a commitment to the multilinear extension of the circuit transcript W. Here P_{arg} crucially relies on streaming access to W to compute the commitment in small-space using Com. (2) P_{arg} and V_{arg} run the protocol Eval in place of all verifier queries to the oracle $\overline{W}$ (line 11, Fig. 6). We state the formal theorem and defer its proof to the full-version.

Theorem 6 (Small-Space Arguments for RAMs). *There exists a public-coin interactive argument for* NP *relations verifiable by time-T space-S random access machines M, in the random oracle model, under the hardness of discrete-log in obliviously sampleable prime-order groups with the following complexity.*

1. *The protocol has perfect completeness, has $O(\log(T))$ rounds and* $\mathrm{polylog}(T)$ *communication, and has witness-extended emulation.*
2. *The prover runs in time $T \cdot \mathrm{polylog}(T)$ and space $S \cdot \mathrm{polylog}(T)$ given input-witness pair $(x; w)$ for M; and*
3. *The verifier runs in time $T \cdot \mathrm{polylog}(T)$ and space* $\mathrm{polylog}(T)$.

We discuss how we modify our interactive argument of knowledge from Theorem 6 to satisfy zero-knowledge and then make the resulting argument non-interactive, thus obtaining Theorem 1.

Zero-Knowledge. We use commit-and-prove techniques introduced in [2,20] and later implemented in [48]. At a high level, this requires making two changes in our base protocols: (1) modify polynomial commitment from Sect. 4 to satisfy zero-knowledge—we modify all commitments sent in both Com and Eval protocols (Fig. 3) to additionally include blinding factors. For example, commitment to $x \in \mathbb{F}$ under generator $g \in \mathbb{G}$ is changed from g^x to $g^x \cdot h^r$ for some randomly sampled $h \xleftarrow{\$} \mathbb{G}$ and $r \xleftarrow{\$} \mathbb{F}$. Further, at the end of the EvalReduce protocol when $N = 1$, prover instead of sending the witness in the clear instead engages with the verifier in Schnorr's zero-knowledge proof of dot-product protocol [43]. This along with hiding of the commitments now ensure that the resulting polynomial commitment is zero-knowledge. (2) We replace all messages sent in the argument Theorem 6 in the clear with Pedersen hiding commitments and use techniques developed in [48] to ensure verifier checks go through. We emphasize that these changes do not asymptotically blow up the complexity of the protocol and, in particular, keep the space-complexity low. Furthermore, this transformation preserves the knowledge-soundness and public-coin features of the underlying argument [48].

Non-interactivity. We apply the Fiat-Shamir (FS) transform [21] to our zero-knowledge argument of knowledge, thereby obtaining a non-interactive, zero-knowledge argument of knowledge. However, note that it is folklore that applying FS to a t-round public-coin argument of knowledge yields a non-interactive argument of knowledge where the extractor runs in time exponential in t. Since our protocol has $O(\log T)$ rounds our extractor runs in $\mathsf{poly}(T)$-time.

Acknowledgements. This work was done in part while Alexander R. Block and Pratik Soni were visiting the FACT Research Center at IDC Herzliya, Israel. Ron Rothblum was supported in part by a Milgrom family grant, by the Israeli Science Foundation (Grants No. 1262/18 and 2137/19), and the Technion Hiroshi Fujiwara cyber security research center and Israel cyber directorate. Alon Rosen is supported in part by ISF grant No. 1399/17 and Project PROMETHEUS (Grant 780701). Pratik Soni was supported in part by NSF grants CNS-1528178, CNS-1929901 and CNS-1936825 (CAREER), Glen and Susanne Culler Chair, ISF grant 1861/16 and AFOSR Award FA9550-17-1-0069. Alexander R. Block was supported in part by NSF grant CCF-1910659.

References

1. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426 (2019). https://eprint.iacr.org/2019/426
2. Ben-Or, M., et al.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 37–56. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_4
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) ICALP 2018. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.14
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23
5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. In: Kleinberg, R.D. (ed.) ITCS 2013, pp. 401–414. ACM (2013). https://doi.org/10.1145/2422436.2422481
6. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2
7. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness. Cryptology ePrint Archive, Report 2019/336 (2019). https://eprint.iacr.org/2019/336
8. Biehl, I., Meyer, B., Wetzel, S.: Ensuring the integrity of agent-based computations by short proofs. In: Rothermel, K., Hohl, F. (eds.) MA 1998. LNCS, vol. 1477, pp. 183–194. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0057658

9. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 111–120. ACM Press (2013). https://doi.org/10.1145/2488608.2488623
10. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. Cryptology ePrint Archive, Report 2012/461 (2012). http://eprint.iacr.org/2012/461
11. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 255–272. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_16
12. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. Cryptology ePrint Archive, Report 2014/846 (2014). http://eprint.iacr.org/2014/846
13. 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, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12
14. Bowe, S., Grigg, J., Hopwood, D.: Halo: recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019). https://eprint.iacr.org/2019/1021
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press (2018). https://doi.org/10.1109/SP.2018.00020
16. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499 (2020). https://eprint.iacr.org/2020/499
17. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_24
18. Chiesa, A., Ojha, D., Spooner, N.: Fractal: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_27
19. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Goldwasser, S. (ed.) ITCS 2012, pp. 90–112. ACM (2012). https://doi.org/10.1145/2090236.2090245
20. Cramer, R., Damgård, I.: Zero-knowledge proofs for finite field arithmetic, or: can zero-knowledge be for free? In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 424–441. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055745
21. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
22. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37

23. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 99–108. ACM Press (2011). https://doi.org/10.1145/1993636.1993651
24. Goldreich, O., Håstad, J.: On the complexity of interactive proofs with bounded communication. Inf. Process. Lett. **67**(4), 205–214 (1998)
25. Goldreich, O., Vadhan, S.P., Wigderson, A.: On interactive proofs with a laconic prover. Comput. Complex. **11**(1–2), 1–53 (2002)
26. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 113–122. ACM Press (2008). https://doi.org/10.1145/1374376.1374396
27. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012
28. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 379–396. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_22
29. Holmgren, J., Rothblum, R.: Delegating computations with (almost) minimal time and space overhead. In: Thorup, M. (ed.) 59th FOCS, pp. 124–135. IEEE Computer Society Press (2018). https://doi.org/10.1109/FOCS.2018.00021
30. Kalai, Y.T., Raz, R.: Probabilistically checkable arguments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 143–159. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_9
31. Kalai, Y.T., Raz, R., Rothblum, R.D.: Delegation for bounded space. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 565–574. ACM Press (2013). https://doi.org/10.1145/2488608.2488679
32. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
33. Kattis, A., Panarin, K., Vlasov, A.: RedShift: transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400 (2019). https://eprint.iacr.org/2019/1400
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press (1992). https://doi.org/10.1145/129712.129782
35. Lindell, Y.: Parallel coin-tossing and constant-round secure two-party computation. J. Cryptol. **16**(3), 143–184 (2003). https://doi.org/10.1007/s00145-002-0143-7
36. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st FOCS, pp. 2–10. IEEE Computer Society Press (1990). https://doi.org/10.1109/FSCS.1990.89518
37. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS, pp. 436–453. IEEE Computer Society Press (1994). https://doi.org/10.1109/SFCS.1994.365746
38. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_13
39. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252. IEEE Computer Society Press (2013). https://doi.org/10.1109/SP.2013.47
40. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9

41. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC, pp. 49–62. ACM Press (2016). https://doi.org/10.1145/2897518.2897652
42. Ron-Zewi, N., Rothblum, R.: Local proofs approaching the witness length. Electron. Colloquium Comput. Complex. **26**, 127 (2019). https://eccc.weizmann.ac.il/report/2019/127
43. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
44. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_25
45. Shamir, A.: IP=PSPACE. In: 31st FOCS, pp. 11–15. IEEE Computer Society Press (1990). https://doi.org/10.1109/FSCS.1990.89519
46. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 71–89. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_5
47. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_1
48. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy, pp. 926–943. IEEE Computer Society Press (2018). https://doi.org/10.1109/SP.2018.00060
49. Wijesekera, P., et al.: The feasibility of dynamically granted permissions: aligning mobile privacy with user preferences. In: 2017 IEEE Symposium on Security and Privacy, pp. 1077–1093. IEEE Computer Society Press (2017). https://doi.org/10.1109/SP.2017.51
50. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy, pp. 859–876. IEEE Computer Society Press (2020). https://doi.org/10.1109/SP40000.2020.00052