Linear-Size Constant-Query IOPs for Delegating Computation

Eli Ben-Sasson[1], Alessandro Chiesa[2], Lior Goldberg[1], Tom Gur[3], Michael Riabzev[1], and Nicholas Spooner[2]

[1] StarkWare
{eli,lior,michael}@starkware.co
[2] UC Berkeley
{alexch,nick.spooner}@berkeley.edu
[3] University of Warwick
tom.gur@warwick.ac.uk

Abstract. We study the problem of delegating computations via interactive proofs that can be probabilistically checked. Known as *interactive oracle proofs* (IOPs), these proofs extend probabilistically checkable proofs (PCPs) to multi-round protocols, and have received much attention due to their application to constructing cryptographic proofs (such as succinct non-interactive arguments). The relevant complexity measures for IOPs in this context are prover and verifier time, and query complexity.

We construct highly efficient IOPs for a rich class of nondeterministic algebraic computations, which includes succinct versions of arithmetic circuit satisfiability and rank-one constraint system (R1CS) satisfiability. For a time-T computation, we obtain prover arithmetic complexity $O(T \log T)$ and verifier complexity $\mathrm{polylog}(T)$. These IOPs are the first to simultaneously achieve the state of the art in prover complexity, due to [14], and in verifier complexity, due to [7]. We also improve upon the query complexity of both schemes.

The efficiency of our prover is a result of our highly optimized proof length; in particular, ours is the first construction that simultaneously achieves linear-size proofs and polylogarithmic-time verification, regardless of query complexity.

Keywords: interactive oracle proofs; probabilistically checkable proofs; delegation of computation

1 Introduction

Verifiable delegation of computation is a central goal in cryptography. The complexity-theoretic study of proof systems has enabled significant progress in this area, and the efficiency of numerous delegation schemes crucially relies on the efficiency of the underlying complexity-theoretic objects.

An influential line of work began with *probabilistically checkable proofs* (PCPs) [5]. These are non-interactive proofs for membership in a language, which admit fast probabilistic verification based on local queries to the proof. While the most prominent application of PCPs is to hardness of approximation [26], seminal works of Kilian [34]

and Micali [37] showed that PCPs can also be used to obtain computationally-sound verifiable delegation schemes that are asymptotically efficient.

The application of PCPs to delegation singles out particular design objectives, distinct from those which arise from hardness of approximation. The relevant complexity measures for PCPs in the context of delegation are: *query complexity*, *verifier time*, and *prover time*. The latter two are self-explanatory, since the proof must be produced and validated; the former arises because in existing delegation schemes based on PCPs, communication complexity depends linearly on the query complexity of a PCP. Note that the running time of the prover is not typically considered in the context of PCPs, because one considers only the existence of a valid PCP and not how it is constructed. For delegation schemes, on the other hand, the time required to generate the proof is often a barrier to practical use.

An ideal PCP for delegation would have *constant* query complexity, *(poly)logarithmic* verifier time and *linear* prover time. State-of-the-art PCPs achieve constant query complexity and polylogarithmic verifier time, but only *quasilinear* ($N \log^c N$) prover time [38]. While the prover time is asymptotically close to optimal, c is a fairly large constant, and more generally the construction uses gap amplification techniques that are not believed to be concretely efficient. The value of c turns out to be very significant in practical settings, but improving it has proven to be a serious challenge in PCP constructions.

In light of these apparent barriers, Ben-Sasson et al. [15] have demonstrated how to obtain computationally-sound delegation schemes from a natural generalization of PCPs known as *interactive oracle proofs* (IOPs) [15, 40]. An IOP is an interactive protocol consisting of multiple rounds, where in each round the verifier sends a challenge and the prover responds with a PCP oracle to which the verifier can make a small number of queries (generalizing the "interactive PCP" model studied in [33]). The proof length of an IOP is the total length of all oracles sent by the prover, and the query complexity is the total number of queries made to these oracles. The study of IOPs explores the tradeoff between a new efficiency measure, round complexity, and other efficiency measures. Viewed in this way, a PCP is an IOP with optimal round complexity.

A recent line of works has demonstrated that this additional freedom is valuable, proving a number of results for IOPs that we do not know how to obtain via PCPs alone [11, 10, 6, 8, 7, 14]. For example, there are constant-round IOPs with linear proof length and constant query complexity for Boolean circuit satisfiability [10], whereas the best linear-size PCPs known have query complexity N^ϵ [18]. Interaction also enables gains in prover time: the FRI protocol [8] is an $O(\log N)$-round IOP of proximity for the Reed–Solomon code where the prover has *linear* arithmetic complexity. In contrast, state-of-the-art PCPs of proximity for this code have quasilinear arithmetic complexity [20, 13]. This theoretical progress has led to IOP-based implementations [7, 14], which are significantly more efficient than those based on PCPs [6].

The work of [15] justifies why exploiting the tradeoff between round complexity and other efficiency measures is advantageous for constructing computationally-sound verifiable delegation schemes. In particular, if we could obtain IOPs with constant round complexity that otherwise match the parameters of an ideal PCP (constant queries, polylogarithmic verifier, linear prover), then we would obtain delegation schemes that

have the same asymptotic efficiency as those derived from an ideal PCP. Thus, for the purposes of verifiable delegation schemes, it suffices to construct such 'ideal' IOPs.

A recent work [14] constructs IOPs for arithmetic circuits with logarithmic query and round complexity where the prover has $O(N \log N)$ (*strictly quasilinear*) arithmetic complexity, and hence bit complexity $\widetilde{O}(N \log N)$. Because the construction emphasizes concrete efficiency over asymptotics, query and round complexity fall somewhat short of the state of the art, but the prover time, while still not linear, is the best among known schemes with subpolynomial query complexity. However, the IOPs in [14] do *not* achieve polylogarithmic verification time: for the language they target even sublinear verification is impossible (without preprocessing) because the size of the input is the same as the size of the computation.

When the arithmetic circuit can be represented succinctly, however, polylogarithmic time verification *is* possible in principle. Unfortunately the protocol of [14] cannot exploit this property, and the verifier would still run in linear time. Our goal is to achieve an *exponential* improvement in this case.

1.1 Our results

In this work we construct IOPs for algebraic computations that are "almost" ideal, namely, we achieve constant query and round complexity, polylogarithmic time for the verifier, and $O(N \log N)$ (*strictly quasilinear*) arithmetic complexity for the prover. Our new IOP protocols match the state-of-the-art prover time of [14], while at the same time achieving an *exponential improvement* in verification time for a rich class of computations. We focus on arithmetic complexity as the natural notion of prover efficiency for IOPs for algebraic problems; moving to bit complexity incurs an additional $\mathrm{poly}(\log \log)$ factor to account for the cost of field multiplication.

While the arithmetic complexity of our prover is not linear, the *length* of the proof is linear in the computation size, which is optimal. The single logarithmic factor in our prover's arithmetic complexity comes *solely from fast Fourier transforms*. In particular, if there were a linear-time encoding procedure for the Reed–Solomon code, our prover would run in linear time, and thereby achieve optimal prover efficiency without any other changes in the scheme itself.

Small fields. All of our results are stated over large fields. Computations over small fields (e.g. $\mathbb{F}_2$) can be handled by moving to an extension field, which introduces an additional logarithmic factor in the proof length and prover time (the same is true of [7, 14]). Even with this additional logarithmic factor, our construction matches the state of the art for prover complexity for succinct *boolean* circuit satisfiability, while improving the verifier running time to polylogarithmic.

Delegating bounded-space algebraic computation Rank-one constraint satisfiability (R1CS) is a natural generalization of arithmetic circuits that is widely used across theoretical and applied constructions of proof systems (see [24]). An R1CS instance is specified by matrices A, B, C over a finite field $\mathbb{F}$, and is satisfied by a vector w if $Aw \circ Bw = Cw$, where $\circ$ is the element-wise (Hadamard) product. Arithmetic circuits reduce in linear-time to R1CS instances.

Many problems of interest, however, involve R1CS instances where the matrices A, B, C have some structure. For example, many applications consider computations that involve checking many Merkle authentication paths — in this case a hash function is invoked many times, within the same path and across different paths. It would be valuable for the verifier to run in time that is related to a *succinct representation* of such instances, rather than to the (much larger) explicit representation that "ignores" the structure. In light of this motivation, we introduce a notion of *succinctly-represented R1CS instances* that capture a rich class of bounded-space algebraic computations. (Later in the paper we refer to these as *algebraic automata*.)

Definition 1 (informal). *A* **succinct R1CS** *instance is specified by matrices $A = [A_0|A_1], B = [B_0|B_1], C = [C_0|C_1] \in \mathbb{F}^{k \times 2k}$ over $\mathbb{F}$, and a time bound T, and is satisfied by a vector $z \in \mathbb{F}^{kT}$ if*

$$\begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \end{pmatrix} w \quad \circ \quad \begin{pmatrix} B_0 & B_1 & & & \\ & B_0 & B_1 & & \\ & & \ddots & \ddots & \\ & & & B_0 & B_1 \end{pmatrix} w \quad = \quad \begin{pmatrix} C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{pmatrix} w$$

The relation Succinct-R1CS *is the set of pairs $(\mathbb{x}, w)$ such that $\mathbb{x}$ is an instance of succinct R1CS which is satisfied by w.*

The size of an *instance* is $O(k^2 + \log T)$, but the size of the computation described is kT. Note that Succinct-R1CS is a PSPACE-complete relation, while the (regular) R1CS relation is merely NP-complete.

To obtain some intuition about the definition, consider the problem of repeated application of an arithmetic circuit $\mathcal{C} \colon \mathbb{F}^n \to \mathbb{F}^n$. Suppose that we want to check that there exists z such that $\mathcal{C}^T(z) = 0^n$, where $\mathcal{C}^T = \mathcal{C}(\mathcal{C}(\cdots \mathcal{C}(\cdot)))$ is the circuit which applies $\mathcal{C}$ iteratively T times. The circuit $\mathcal{C}^T$ has size $\Omega(|\mathcal{C}| \cdot T)$, and if the verifier were to "unroll" the circuit then it would pay this cost in time. However, the R1CS instance corresponding to $\mathcal{C}^T$ is of the above form, with $k = O(|\mathcal{C}|)$, where (roughly) the matrices A_0, B_0, C_0 represent the gates of $\mathcal{C}$ and A_1, B_1, C_1 represent the wires between adjacent copies of $\mathcal{C}$. (The condition that the output of $\mathcal{C}^T$ is zero is encoded separately as a "boundary constraint".)

Our first result gives a constant-round IOP for satisfiability of succinct R1CS where the verifier runs in time $\mathrm{poly}(k, \log T)$, the prover has arithmetic complexity $O(kT \log kT)$, and the proof length is $O(kT \log |\mathbb{F}|)$ (linear in the computation transcript). In the theorem statement below we take $k = O(1)$ for simplicity.

Theorem 1 (informal). *There is a universal constant $\epsilon_0 \in (0, 1)$ such that, for any computation time bound $T(n)$ and large smooth field family $\mathbb{F}(n)$, there is a 4-round IOP for succinct R1CS over $\mathbb{F}(n)$, with proof length $O(T(n) \log |\mathbb{F}(n)|)$, 4 queries, and soundness error ϵ_0. The prover uses $O(T(n) \log T(n))$ field operations and the verifier uses $\mathrm{poly}(n, \log T(n))$ field operations.*

As in prior work (e.g., [20]), "large smooth field" refers to a field of size $\Omega(N)$, whose additive or multiplicative group has a nice decomposition. For example, ensembles

of large enough binary fields have this property, as well as prime fields with smooth multiplicative groups.

Delegating unbounded-space algebraic computation While algebraic automata capture a useful class of computations, they are restricted to space-bounded computation (recall $\mathsf{Succinct\text{-}R1CS} \in \mathsf{PSPACE}$). In particular, using Theorem 1 we can obtain useful delegation protocols for computations whose space usage is much smaller than their running time.

To handle computations which use more space, we introduce the *algebraic machine* relation. This is a natural algebraic analogue of the bounded accepting problem for nondeterminstic random-access machines, where the transition function is an arithmetic (rather than Boolean) circuit. It is NEXP-complete via a linear time reduction from succinct arithmetic circuit satisfiability.

Theorem 2 (informal). *There is a universal constant $\epsilon_0 \in (0,1)$ such that, for any computation time bound $T(n)$ and large smooth field $\mathbb{F}(n)$, there is a 5-round IOP for the satisfiability problem of $T(n)$-time algebraic machines over $\mathbb{F}(n)$, with proof length $O(T(n)\log|\mathbb{F}(n)|)$, 5 queries, and soundness error ϵ_0. The prover uses $O(T(n)\log T(n))$ field operations and the verifier uses $\mathrm{poly}(n, \log T(n))$ field operations.*

For simplicity, as with Theorem 1 we have stated Theorem 2 for machines whose description is a constant number of field elements, or $\Theta(\log|\mathbb{F}|)$ bits. The proof length is *linear* in the size of the computation trace, which is $N := \Theta(T\log|\mathbb{F}|)$ bits. We stress that the number of queries is 5, *regardless* of the choice of machine.

The above theorem is obtained by bootstrapping Theorem 1. Namely we show that leveraging interaction, we can design an automaton which checks whether a pair of automata have satisfying assignments which are permutations of one another; for more details see Section 2.4.

On the power of machines. In the linear-length regime, the choice of computational model supported by a proof protocol is important, because reductions between problems typically introduce logarithmic factors. For example, it is not known how to reduce a random-access machine, or even a Turing machine, to a circuit of linear size. Indeed, the sublinear-query PCP of [18] achieves linear proof size for circuits but not machine computations. We thus view Theorem 2 as particularly appealing, because it achieves linear length for a powerful model of computation, algebraic machines, which facilitates linear-size reductions from many other problems; notably, succinct arithmetic circuit satisfiability. We view the identification of a model which is both highly expressive and amenable to efficient probabilistic checking using IOPs as a contribution of this work.

1.2 Relation to prior work

There are relatively few works which explicitly deal with prover complexity for PCP and IOP constructions. We present a comparison of the relevant parameters for each construction in Table 1. Since we are concerned with logarithmic factors, it is not sufficient to specify only a complexity class (NP or NEXP) for each one. Instead,

for each proof system we give a canonical expressive language for which the given parameters are achieved. In particular, the first three proof systems are for boolean circuit problems, and the latter three are for arithmetic circuit problems. For purposes of comparison, all of the parameters for both boolean and arithmetic constructions are presented in terms of bit complexity.

	rounds	**circuit type**	**prover time**	**verifier time**	**proof length**	**queries**
[38]	1	succinct boolean	$N\,\mathrm{polylog}(N)$ *	$\mathrm{polylog}(N)$	$N\,\mathrm{polylog}(N)$	$O(1)$
[18]	1	boolean	$\mathrm{poly}(N)$ †	$\mathrm{poly}(N)$ †	$O_\epsilon(N)$	N^ϵ
[10]	3	boolean	$\mathrm{poly}(N)$	$\mathrm{poly}(N)$	$O(N)$	$O(1)$
[7]	$O(\log N)$	succinct arithmetic ◇	$\widetilde{O}(N \log^2 N)$ ‡	$\mathrm{polylog}(N)$	$O(N \log N)$	$O(\log N)$
[14]	$O(\log N)$	arithmetic ◇	$\widetilde{O}(N \log N)$ ‡	$\mathrm{poly}(N)$	$O(N)$	$O(\log N)$
this work	5	succinct arithmetic ◇	$\widetilde{O}(N \log N)$ ‡	$\mathrm{polylog}(N)$	$O(N)$	5

Table 1: Comparison of PCP/IOP constructions for circuit satisfiability problems for a (fixed) constant soundness. Here N is the size of the circuit *in bits*, which means that for ASAT and Succinct-ASAT, N implicitly includes a factor of $\log |\mathbb{F}|$. For succinct problems, the circuit size N is exponential in the size of its description.
*: [38] shows a $\mathrm{poly}(N)$ bound; this tighter bound is due to [13].
◇: The size of the underlying field must grow as $\Omega(N)$ to achieve the stated efficiency. Problems over smaller fields (e.g. boolean circuits) incur a multiplicative cost of $\log N$ in prover time and proof length.
†: The specified time is for *non-uniform* computation (each input size receives $\mathrm{poly}(N)$ advice bits).
‡: The notation $\widetilde{O}$ hides $\mathrm{poly}(\log \log N)$ factors, which arise because here we consider the *bit complexity* of the prover (rather than the arithmetic complexity).

From a technical perspective, prover running time is tightly connected to proof length, which is more well-studied. In all constructions, the proof length is a lower bound on the prover running time. Moreover, in the most prover-efficient constructions [7, 14], the dominant cost for the prover is in computing Reed–Solomon encodings, which means that for proof length ℓ the arithmetic complexity of the prover is $O(\ell \log \ell)$. Finally, since proof length is an information-theoretic property of the system, it is also usually easier to analyse. We proceed by discussing the state of the art for PCPs and IOPs with *linear* (optimal) proof length, since this is what our construction achieves.

There are two natural approaches that one could follow to simultaneously achieve linear proof length and constant query complexity: (1) start from a construction with constant query complexity and reduce proof length; or (2) start from a construction with linear proof length and reduce query complexity. We summarize prior works that have followed these approaches, and highlight the limitations that arise in each case.

Approach (1). The first approach has been studied extensively [5, 39, 31, 21, 27, 17], leading to PCPs for NEXP with proof length $N\,\mathrm{polylog}(N)$ and query complexity $O(1)$

[20, 16, 25, 38]. Later works have reduced the logarithmic factors in the proof length [12, 13], but attempts to achieve linear length have failed. Recent work has obtained IOPs with proof length $O(N \log N)$ but at the cost of increasing query complexity from $O(1)$ to $O(\log N)$ [6, 7].

Approach (2). The second approach has received much less attention. Insisting on linear proof length significantly restricts the available techniques because many tools introduce logarithmic factors in proof length. For example, one cannot rely on arithmetization via multivariate polynomials and standard low-degree tests, nor rely on algebraic embeddings via de Bruijn graphs for routing; in addition, query-reduction techniques for interactive PCPs [33] do not apply to the linear proof length regime. The state-of-the-art in linear-length PCPs is due to [18], and the construction is based on a non-uniform family of algebraic geometry (AG) codes (every input size needs a polynomial-size advice string). In more detail, [18] proves that for every $\epsilon \in (0, 1)$ there is a (non-uniform) PCP for the NP-complete problem CSAT (Boolean circuit satisfiability) with proof length $2^{O(1/\epsilon)}N$ and query complexity N^{ϵ}, much more than our goal of $O(1)$.

By leveraging interaction, [10] obtains IOPs for CSAT with proof length $O(N)$ and query complexity $O(1)$. This is a natural starting point for our goal of achieving polylogarithmic-time verification, because we are "only" left to extend this result from CSAT to its succinct analogue, Succinct-CSAT. Unfortunately, the construction in [10] uses AG codes and such an extension would, in particular, require obtaining a succinct representation of a dense asymptotically good family of AG codes over a small field, which is out of reach of current techniques. More generally, we do not know of any suitable code over small fields, which currently seems to prevent us from obtaining linear-size IOPs for Succinct-CSAT. Moreover, obtaining an efficient prover would require efficient encoding and decoding procedures for AG codes.

We now consider arithmetic circuit satisfiability (ASAT) defined over fields $\mathbb{F}$ that are large (of size $\Omega(N)$). In this regime, [14] obtains IOPs for ASAT with proof length $O(N)$ and query complexity $O(\log N)$. The arithmetization, following [20], is based on the Reed–Solomon code and uses the algebraic structure of large smooth fields. Testing is done via FRI [8], a recent IOP of proximity for the Reed–Solomon code with linear proof length and logarithmic query complexity. The construction in [14], which we will build upon, falls short of our goal on two fronts: verification is linear in the size of the circuit rather than polylogarithmic, and query complexity is logarithmic rather than constant.

1.3 Open questions

We highlight four problems left open by our work.

Optimal arithmetic complexity. The prover in our construction has strictly quasilinear arithmetic complexity and produces a proof of linear size. A natural question is whether the arithmetic complexity of the prover can be reduced to linear. To do so with our construction would require a breakthrough in encoding algorithms for the Reed–Solomon code. A promising direction is to build IOPs based on codes with linear-time encoding procedures [42, 30, 23].

All fields. The question of whether it is possible to simultaneously achieve linear-length proofs and polylogarithmic-time verifier for Succinct-ASAT over *any* field $\mathbb{F}$ remains open. Progress on this question motivates the search for arithmetization-friendly families of good codes beyond the Reed–Solomon code. For example, the case of $\mathbb{F} = \mathbb{F}_2$, which corresponds to boolean circuits, motivates the search for succinctly-represented families of good algebraic-geometry codes over constant-size fields with fast encoding algorithms.

Zero knowledge. Zero knowledge, while not a goal of this work, is a desirable property, because zero knowledge PCP/IOPs lead to zero knowledge succinct arguments [32, 15]. Straightforward modifications to the protocol, similar to [14], achieve a notion of zero knowledge wherein the simulator runs in time polynomial in the size of the computation being checked, which is meaningful for nondeterministic problems since it does not have access to the witness.

There is a stronger notion of zero knowledge for succinct languages where the simulator runs in *polylogarithmic* time, which is exponentially more efficient. This gap was precisely the subject of a work on designing succinct simulators for certain tests [9]. Whether succinct simulators can be designed for low-degree tests that we could use for our protocol remains an intriguing problem that we leave to future research.

Round complexity. Our protocol has 5 rounds. Round complexity can be reduced to 4 at the cost of increased (constant) query complexity. Reducing round complexity beyond this while preserving linear proof length and polylogarithmic time verification, or finding evidence against this possibility, remains open.

2 Technical overview

We discuss the main ideas behind our results. Our goal is to construct an IOP for algebraic machines, over a large field $\mathbb{F}$, with prover arithmetic complexity which is *strictly quasilinear* in the size of the computation (i.e. $O(N \log N)$); and crucially, the running time of the verifier is *polylogarithmic* in the size of the computation (more precisely, polynomial in the machine description). Additionally, we strive to optimize the query and round complexity of this IOP. We stress that no prior work achieves non-trivial strictly quasilinear prover PCPs or IOPs wherein the verifier runs in polylogarithmic time in the size of the computation.

Following [7, 14], our construction relies heavily on the Reed–Solomon code, and the dominant cost for the prover is in the encoding procedure. Thus to achieve strictly quasilinear arithmetic complexity in our construction, we *must* achieve a linear proof length. Thus from this point on, discussion will focus primarily on proof length.

The rest of this section is organized as follows. In Section 2.1 we discuss our starting point, which is a construction of [14]. In Section 2.2 we discuss our approach to overcoming the limitations of prior work by describing a new protocol for checking succinctly-represented linear relations; this achieves an *exponential* improvement over the prior state of the art. In Section 2.3 we discuss how to overcome the challenges that arise when attempting to build on this exponential improvement to checking the computation of algebraic automata. In Section 2.4, we discuss how to extend these techniques to algebraic machines (which capture succinct arithmetic circuit satisfiability).

In Section 2.5 we describe a modular framework, which we call *oracle reductions*, in which we prove our results.

2.1 Our starting point

The starting point of our work is [14], which obtains IOPs for R1CS with proof length $O(N)$ and query complexity $O(\log N)$, and in which the prover uses $O(N \log N)$ field operations and the verifier uses $O(N)$. Recall that the R1CS problem is as follows: given matrices A, B, C over a finite field $\mathbb{F}$, the problem asks whether there exists a witness vector w, where some entries are fixed to known values, for which the following ***R1CS equation*** holds: $(Aw) \circ (Bw) = Cw$, where "$\circ$" denotes entry-wise product.

Our initial goal in this paper is to achieve an IOP for satisfiability of algebraic automata. This entails an *exponential* improvement in the running time of the verifier, from linear in the circuit size to polylogarithmic in the circuit size. Moreover, we need to achieve this improvement with proof length $O(N)$ and query complexity $O(1)$ (and a prover that uses $O(N \log N)$ field operations).

The ideas behind our results are better understood if we first briefly recall the IOP of [14]. The prover sends to the verifier four oracles $\pi_w, \pi_A, \pi_B, \pi_C$ that are purported encodings of w, Aw, Bw, Cw. The verifier must now check two sub-problems: (a) if π_w encodes w then π_A, π_B, π_C respectively encode Aw, Bw, Cw; and (b) if π_A, π_B, π_C encode w_A, w_B, w_C then $w_A \circ w_B = w_C$.

As usual, there is a tension in selecting the encoding used to obtain the oracles $\pi_w, \pi_A, \pi_B, \pi_C$. One needs an encoding that allows for non-trivial checking protocols, e.g., where the verifier makes a small number of queries. On the other hand, the encoding must have constant rate so that proof length can be linear.

The encoding used relies on univariate polynomials: denote by $\mathrm{RS}[L, \rho] \subseteq \mathbb{F}^L$ the Reed-Solomon code over a subset L of a field $\mathbb{F}$ with rate parameter $\rho \in (0, 1]$ (that is, the set of all functions $f\colon L \to \mathbb{F}$ of degree less than $\rho|L|$). Also, denote by $\hat{f}$ the unique univariate polynomial of degree less than $\rho|L|$ whose evaluation on L equals f. Given a subset $H \subseteq \mathbb{F}$ (the *domain* of the encoding), a Reed–Solomon codeword f *encodes* $x \in \mathbb{F}^H$ if $\hat{f}(a) = x_a$ for all $a \in H$; for each x, there is a unique encoding f_x of x of minimal rate. We can now restate the aforementioned sub-problems in terms of the Reed–Solomon code.

- **Lincheck:** given a subset $H \subseteq \mathbb{F}$, Reed–Solomon codewords $f, g \in \mathrm{RS}[L, \rho]$ that encode $x, y \in \mathbb{F}^H$ respectively, and a matrix $M \in \mathbb{F}^{H \times H}$, check that $x = My$.
- **Rowcheck:** given a subset $H \subseteq \mathbb{F}$ and Reed–Solomon codewords $f, g, h \in \mathrm{RS}[L, \rho]$ that encode $x, y, z \in \mathbb{F}^H$ respectively, check that $x \circ y = z$.

The IOP in [14] is obtained by combining sub-protocols for these sub-problems, a *lincheck protocol* and a *rowcheck protocol*. The latter is a simple reduction from the rowcheck problem to testing membership in the Reed–Solomon code, and is implied by standard PCP tools. The former, however, is a novel (and non-trivial) reduction from the lincheck problem to testing membership in the Reed–Solomon code.

While on the one hand the verifier in the rowcheck protocol runs in time that is polylogarithmic in $|H|$ (which is good) the verifier in the lincheck protocol runs in time

that is linear in $|H|$ (which is much too slow). In other words, if we simply invoked the IOP in [14] on the circuit described by a succinct R1CS instance, the verifier would run in time that is linear in T, which is exponentially worse than our goal of $\mathrm{polylog}(T)$. This state of affairs in the starting point of our work.

Next, in Section 2.2, we discuss how to obtain a succinct lincheck protocol that, for suitable linear relations, is exponentially more efficient. After that, in Section 2.3, we discuss the notion of algebraic automata in detail and describe how the succinct lincheck protocol enables efficient probabilistic checking of algebraic automata. Finally in Section 2.4 we describe how we can bootstrap our protocol for algebraic automata to check the more powerful algebraic machine model.

Throughout, we present our contributions as *oracle reductions* from some computational task to testing membership in the Reed–Solomon code. Loosely speaking, these are reductions in the setting of the IOP model (and therefore, in particular, allow interaction in which the prover sends PCP oracles). This abstraction allows us to decouple IOP protocol-design from the low-degree test that we invoke at the end of the protocol. See Section 2.5 for details.

2.2 Checking succinctly-represented linear relations

Following the above discussion, we now temporarily restrict our attention to devising a lincheck protocol, which reduces checking linear relations defined by matrices $M \in \mathbb{F}^{H \times H}$ to testing membership in the Reed–Solomon code, in which the verifier runs in time that is *polylogarithmic* in $|H|$. This is not possible in general, however, because the verifier needs to at least *read the description* of the matrix M. We shall have to consider matrices M that have a special structure that can be exploited to obtain an exponential improvement in verifier time. This improvement is a core technical contribution of this paper, and we refer to the resulting reduction as the *succinct lincheck protocol*. We start by describing the ideas behind the (non-succinct) lincheck protocol of [14].

Definition 2 (informal). *In the lincheck problem, we are given a subset $H \subseteq \mathbb{F}$, Reed–Solomon codewords $f, g \in \mathrm{RS}[L, \rho]$ encoding vectors $x, y \in \mathbb{F}^H$, and a matrix $M \in \mathbb{F}^{H \times H}$. The goal is to check that $x = My$.*

A simple probabilistic test for the claim "$x = My$" is to check that $\langle r, x - My \rangle = 0$ for a random $r \in \mathbb{F}^H$. Indeed, if $x \neq My$, then $\Pr_{r \in \mathbb{F}^H}[\langle r, x - My \rangle = 0] = 1/|\mathbb{F}|$. However, this approach would require the verifier to sample, and send to the prover, $|H|$ random field elements (too many).

A natural derandomization is to choose $\boldsymbol{r}$ using a small-bias generator over $\mathbb{F}$, rather than uniformly at random. A small-bias generator G over $\mathbb{F}$ is a function with the property that for any nonzero $z \in \mathbb{F}^H$, it holds with high probability over $\rho \in \{0,1\}^\ell$ that $\langle z, G(\rho) \rangle \neq 0$. Now the verifier needs to send only ℓ bits to the prover, which can be much smaller than $|H| \log |\mathbb{F}|$.

A natural choice (used also, e.g., in [5, §5.2]) is the powering construction of [1], which requires sending a *single* random field element ($\ell = \log |\mathbb{F}|$), and incurs only a modest increase in soundness error. In this construction, we define a vector $\boldsymbol{r}(X) \in \mathbb{F}[X]^H$ of linearly independent polynomials in X, given by $(1, X, X^2, \ldots, X^{|H|-1})$.

The small-bias generator is then $G(\alpha) := \boldsymbol{r}(\alpha)$ for $\alpha \in \mathbb{F}$. If z is nonzero then $h(X) := \langle \boldsymbol{r}(X), z\rangle$ is a nonzero polynomial and so $\Pr_{\alpha\in\mathbb{F}}[\langle G(\alpha), z\rangle = 0] \leq \deg(h)/|\mathbb{F}|$. The verifier now merely has to sample and send $\alpha \in \mathbb{F}$, and the prover must then prove the claim "$h(\alpha) = 0$" to the verifier. Rearranging, this is the same as testing that $\langle \boldsymbol{r}(\alpha), x\rangle - \langle \boldsymbol{r}(\alpha)M, y\rangle = 0$. The problem is thus reduced to checking inner products of known vectors with oracles.

In the setting of Reed–Solomon codewords, if f_u is an encoding of u and f_v is an encoding of v, then $f_u \cdot f_v$ is an encoding of $u \circ v$, the pointwise product of u and v. Hence, to check that $\langle u, v\rangle = c$, it suffices to check that the low-degree polynomial $f_u \cdot f_v$ sums to c on H, since $\langle u, v\rangle = \sum_{h\in H} f_u(h)f_v(h)$. This can be achieved by running the *univariate sumcheck protocol* ([14]) on the codeword $f_u \cdot f_v$. This protocol requires the verifier to efficiently determine the value of $f_u \cdot f_v$ at a given point in L.

The inefficiency. The foregoing discussion tells us that, to solve the lincheck problem, the verifier must determine the value of the Reed–Solomon encodings of $\boldsymbol{r}(\alpha) \circ x$ and $\boldsymbol{r}(\alpha)M \circ y$ at a given point in L. The encodings of the vectors x and y are provided (as f and g). Hence it suffices for the verifier to evaluate *low-degree extensions* of $\boldsymbol{r}(\alpha)$ and $\boldsymbol{r}(\alpha)M$ at a given point, and then perform a field multiplication.

This last step is the computational bottleneck of the protocol. In [14], the verifier evaluates the low-degree extensions of $\boldsymbol{r}(\alpha)$ and $M^\top \boldsymbol{r}(\alpha)$ via Lagrange interpolation, which requires time $\Omega(|H|)$. To make our verifier efficient, we must evaluate both low-degree extensions in time $\mathrm{polylog}(|H|)$. In particular, this requires that M be succinctly represented, since computing the low-degree extension of $\boldsymbol{r}(\alpha)M$ in general requires time linear in the number of nonzero entries in M, which is at least $|H|$.

The lincheck protocol in [14] chooses the linearly independent polynomials $\boldsymbol{r}(X)$ to be the *standard* (or *coefficient*) basis $(1, X, \ldots, X^{|H|-1})$. For this basis, however, we do not know how to efficiently evaluate the low-degree extension of $\boldsymbol{r}(\alpha)$. This problem must be addressed *regardless* of the matrix M.

A new basis and succinct matrices. We leverage certain algebraic properties to overcome the above problem. There is another natural choice of basis for polynomials, the *Lagrange basis* $(L_{H,h}(X))_{h\in H}$, where $L_{H,h}$ is the unique polynomial of degree less than $|H|$ with $L_{H,h}(h) = 1$ and $L_{H,\alpha}(\gamma) = 0$ for all $\gamma \in H \setminus \{h\}$. We observe that the low-degree extension of $\boldsymbol{r}(\alpha) = (L_{H,h}(\alpha))_{h\in H} \in \mathbb{F}^H$ has a simple form that allows one to evaluate it in time $\mathrm{polylog}(|H|)$ provided that H is an additive or multiplicative subgroup of $\mathbb{F}$. In other words, the Lagrange basis yields a small-bias generator over $\mathbb{F}$ whose low-degree extension can be computed efficiently.

It remains to find a useful class of succinctly-represented matrices M for which one can efficiently evaluate a low-degree extension of $\boldsymbol{r}(\alpha)M \in \mathbb{F}^H$. The foregoing discussion suggests a natural condition: *if* we can efficiently compute a low-degree extension of a vector $v \in \mathbb{F}^H$ *then* we should also be able to efficiently compute a low-degree extension of the vector vM. If this holds for all vectors v, we say that the matrix M is (algebraically) *succinct*. For example, the identity matrix satisfies this definition (trivially), and so does the matrix with 1s on the superdiagonal for appropriate choices of $\mathbb{F}$ and H.

In sum, if we choose the Lagrange basis in the lincheck protocol and the linear relation is specified by a succinct matrix, then, with some work, we obtain a lincheck pro-

tocol where the verifier runs in time $\mathrm{polylog}(|H|)$. To check satisfiability of succinctly-represented arithmetic circuits, however, we need to handle a more general class of matrices, described next.

Succinct lincheck for semisuccinct matrices. We will relax the condition on a matrix M in a way that captures the matrices that arise when checking succinctly-described arithmetic circuits, while still allowing us to obtain a lincheck protocol in which the verifier runs in time $\mathrm{polylog}(|H|)$.

We show that the matrices that we consider are *semisuccinct*, namely, they can be decomposed into a "large" part that is succinct and a "small" part that has no special structure.[4] This structure should appear familiar, because it is analogous to how a succinctly-described circuit consists of a small arbitrary component (the circuit descriptor) that is repeatedly used in a structured way to define the large circuit. Another analogy is how in an automaton or machine computation a small, and arbitrary, transition function is repeatedly applied across a large computation.

Specifically, by "decompose" we mean that the matrix $M \in \mathbb{F}^{H \times H}$ can be written as the *Kronecker product* of a succinct matrix $A \in \mathbb{F}^{H_1 \times H_1}$ and a small matrix $B \in \mathbb{F}^{H_2 \times H_2}$; we write $M = A \otimes B$. (Succinctly representing a large operator like M via the tensor product of simpler operators should be a natural idea to readers familiar with quantum information.) In order for the product to be well-defined, we must supply a bijection $\Phi\colon H \to H_1 \times H_2$. If this bijection satisfies certain algebraic properties, which preserve the succinctness of the matrix A, we call it a *bivariate embedding*.

We obtain a succinct lincheck protocol for semisuccinct matrices.

Lemma 1 (informal). *There is a linear-length reduction from the lincheck problem for semisuccinct linear relations to testing membership in the Reed–Solomon code, where the verifier runs in polylogarithmic time.*

Next, we discuss how to obtain a reduction from algebraic automata (succinct R1CS) to testing membership in the Reed–Solomon code, where the verifier runs in time that is *polylogarithmic* in the circuit size, by building on our succinct lincheck protocol for semisuccinct matrices.

2.3 Checking bounded-space computations in polylogarithmic time

An instance of the *algebraic automaton* relation is specified by three $k \times 2k$ matrices (A, B, C) over $\mathbb{F}$, and a time bound T. A witness $f\colon [T] \to \mathbb{F}^k$ is valid if

$$\forall\, t \in [T-1] \quad Af(t, t+1) \circ Bf(t, t+1) = Cf(t, t+1) \ , \tag{1}$$

where $f(t, t+1) \coloneqq f(t) \| f(t+1)$ is the concatenation of the consecutive states $f(t) \in \mathbb{F}^k$ and $f(t+1) \in \mathbb{F}^k$.

We use the term "algebraic automata" since one can think of A, B, C as specifying the transition relation of a computational device with k algebraic registers, and f as an *execution trace* specifying an accepting computation of the device. The relation

[4] We actually need to handle matrices that are the *sum* of semisuccinct matrices, but we ignore this in this high-level discussion.

is PSPACE-complete: it is in NPSPACE because it can be checked by a polynomial-space Turing machine with one-directional access to an exponential-size witness, and recall that NPSPACE = PSPACE; also, it is PSPACE-hard because given an arithmetic circuit specifying the transition relation of a polynomial-space machine, we can find an equisatisfiable R1CS instance in linear time.

If we view the execution trace f as a vector $f = f(1)\| \cdots \|f(T) \in \mathbb{F}^{Tk}$, then we can rewrite the condition in Eq. (1) via the following (possibly exponentially large) R1CS equation:

$$\begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \end{pmatrix} f \quad \circ \quad \begin{pmatrix} B_0 & B_1 & & & \\ & B_0 & B_1 & & \\ & & \ddots & \ddots & \\ & & & B_0 & B_1 \end{pmatrix} f \quad = \quad \begin{pmatrix} C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{pmatrix} f$$

where $A_0, A_1 \in \mathbb{F}^{k \times k}$ are the first half and second half of A respectively; and likewise for B and C. We thus see that algebraic automata are equivalent to Succinct-R1CS.

The matrices in the above R1CS equation have a rigid block structure that we refer to as a *staircase*. Given the discussions in Sections 2.1 and 2.2, in order to achieve polylogarithmic verification time, *it suffices to show that staircase matrices are semisuccinct* (or, at least, the sum of few semisuccinct matrices).

So let $S(M_0, M_1)$ be the staircase matrix of two given $k \times k$ matrices M_0, M_1 over $\mathbb{F}$. Namely, $S(M_0, M_1)$ is the $Tk \times Tk$ matrix with M_0-blocks on the diagonal and M_1-blocks on the superdiagonal. Observe that:

1. we can write the matrix with M_0-blocks on the diagonal as $I \otimes M_0$, where I is the $T \times T$ identity matrix;
2. we can write the matrix with M_1-blocks on the superdiagonal as $I^{\rightarrow} \otimes M_1$, where $I^{\rightarrow}$ is the $T \times T$ matrix with 1s on the superdiagonal.

Under an appropriate mapping from $[Tk]$ into a subset of $\mathbb{F}$, we prove that both of these matrices are semisuccinct. This tells us that $S(M_0, M_1)$ is the sum of two semisuccinct matrices:

$$S(M_0, M_1) := \begin{pmatrix} M_0 & M_1 & & & \\ & M_0 & M_1 & & \\ & & \ddots & \ddots & \\ & & & M_0 & M_1 \\ & & & & M_0 \end{pmatrix} = I \otimes M_0 + I^{\rightarrow} \otimes M_1 \in \mathbb{F}^{Tk} \times \mathbb{F}^{Tk} \ .$$

(Note that the above is not exactly the matrix structure we want, because of the extra M_0 block; we handle this technicality separately.) We obtain the following lemma.

Lemma 2 (informal). *There is a linear-length reduction from the algebraic automaton relation to testing membership in the Reed–Solomon code, where the verifier runs in time* $\mathrm{poly}(k, \log T)$.

2.4 Checking machine computations in polylogarithmic time

An instance of the *algebraic (R1CS) machine* relation is specified by two algebraic (R1CS) automata $(\mathcal{A}, \mathcal{A}')$. A witness (f, π), where $f\colon [T] \to \mathbb{F}^k$ is an execution trace

and $\pi\colon [T] \to [T]$ is a permutation, is valid if: (1) f is a valid witness for the automaton $\mathcal{A}$, and (2) $f \circ \pi$ is a valid witness for the automaton $\mathcal{A}'$. The algebraic machine relation is NEXP-complete, as the NEXP-complete problem of succinct arithmetic circuit satisfiability reduces to it in linear time.

Execution traces for machines. Before we discuss how we reduce from the algebraic machine relation, we briefly explain why the above relation is a natural problem to consider, and in particular why it has anything to do with (random-access) machines. Recall that a random-access machine is specified by a list of instructions, each of which is an arithmetic operation, a control-flow operation, or a read/write to memory. One way to represent the execution trace for the machine is to record the state of the entire memory at each time step; for a time-T space-S computation, such an execution trace has size $\Theta(TS)$ (much more than linear!). Yet, the machine can access only a *single* memory location at each time step. Thus, instead of writing down the state of the entire memory at each time step, we could hope to only write the state of the accessed address — this would reduce the size of the trace to $\Theta(T \log S)$. The problem then is that it is no longer possible to check consistency of memory via local constraints because the same address can be accessed at any time.

Classical techniques of Gurevich and Shelah [29] tell us that one can efficiently represent an execution trace for a machine via *two* execution traces that are *permutations of one another*. Informally, sorting the execution trace by time enables us to check the transition relation of the machine; and sorting the execution trace by the accessed addressed (and then by time) enables us to locally check that memory is consistent. (One must ensure that, for each location, if we write some value to memory and then read the same address, we retrieve that same value.) The transition relation and memory consistency can each be expressed individually as automata. This view of machines immediately gives rise to the algebraic machine relation above.

Checking the algebraic machine relation. We have discussed how to check automata in Section 2.3, so it remains to check that the traces are permutations of one another. Historically this has been achieved in the PCP literature using algebraic embeddings of routing networks; e.g., see [12]. The problem is that this increases the size of the representation of the execution trace by at least a logarithmic factor. We instead use an interactive permutation test from the program checking literature [36, 22]. The test is based on the observation that $u \in \mathbb{F}^T$ is a permutation of $v \in \mathbb{F}^T$ if and only if the multi-sets given by their elements are equal, which is true if and only if the polynomials $p_u(X) = \prod_{i=1}^{T}(X - u_i)$ and $p_v(X) = \prod_{i=1}^{T}(X - v_i)$ are equal. Thus it suffices to evaluate each polynomial at a random point and check equality.

These polynomials require time $\Theta(T)$ to evaluate, which in our setting is exponential. Therefore the prover must assist the verifier with the evaluation. We show that evaluating this polynomial can be expressed as an algebraic automaton, and can therefore be checked again using the protocol from Section 2.3.

The reader may believe that by now we have reduced checking an algebraic machine to checking three instances of algebraic automata. Recall, however, that the algebraic automaton relation is PSPACE-complete, whereas the algebraic machine relation is NEXP-complete. What happened? The answer lies in the randomness used in the permutation automaton. In order to check that u is a permutation of v, the prover must first

commit to u and v before the verifier chooses his evaluation point α, and then the prover sends the trace of the automaton that evaluates $p_u(\alpha), p_v(\alpha)$. This trace depends on the choice of α, and so we use interaction. This is captured by the *interactive automaton relation*, which is NEXP-complete; it can be checked in essentially the same way as the automaton relation.

We hence obtain the following lemma.

Lemma 3 (informal). *There is a linear-length reduction from the algebraic machine relation to testing membership in the Reed–Solomon code, where the verifier runs in time* $\mathrm{poly}(k, \log T)$.

2.5 Oracle reductions

Many results in this paper describe IOPs that reduce a computational problem to membership in (a subcode of) the Reed–Solomon code. We find it useful to capture this class of reductions via a precise definition. This lets us prove general lemmas about such reductions, and obtain our protocols in a modular fashion.

We thus formulate a new notion that we call *interactive oracle reductions* (in short, *oracle reductions*). Informally, an oracle reduction is a protocol that reduces from a computational problem to testing membership in a code (in this paper, the code is the interleaved Reed–Solomon code). This is a well-understood idea in constructions of PCPs and IOPs. Our contribution is to provide a formal framework for this technique.

We illustrate the notion of oracle reductions via an example. Consider the problem of testing proximity to the *vanishing Reed–Solomon code*, which plays an important role in a PCP of Ben-Sasson and Sudan [20] and several other PCPs/IOPs. Informally, the goal is to test whether a univariate polynomial f, provided as an oracle, is zero everywhere on a subset H of $\mathbb{F}$.

We describe an oracle reduction that maps the foregoing problem to the problem of testing membership in the Reed–Solomon code of the related polynomial $g \coloneqq f/\mathbb{Z}_H$. Observe that f is divisible by $\mathbb{Z}_H$ if and only if f is zero everywhere in H, and so g is in the Reed–Solomon code if and only if f satisfies the desired property. But what exactly *is* g? In the oracle reduction framework, we refer to g as a *virtual oracle*: an oracle whose value at any given point in its domain can be determined efficiently by making a small number of queries to concrete oracles. In this case, so long as the domain L we choose for g does not intersect H, a verifier can evaluate g at any point $\alpha \in L$ with only a single query to f. To test that g is low degree, the verifier can invoke any low-degree test on g, and simulate queries to the virtual oracle g via queries to f.

The two main parameters in an oracle reduction are the *proof length*, which is simply the total length of the oracles sent by the prover, and the *locality*, which is the number of queries one would have to make to the concrete oracles to answer a single query to any virtual oracle (in this paper, locality always equals the number of rounds). Using the perspective of oracle reductions, our main theorems (Theorems 1 and 2) follow by combining two main sub-components: (1) a linear-length 3-local oracle reduction from the algebraic automata or machine problem to proximity testing to the Reed–Solomon code (discussed in Sections 2.3 and 2.4); and (2) a linear-length strictly quasilinear prover 3-query IOP for testing proximity to the Reed–Solomon code from [10].

3 Roadmap

Fig. 1 below provides a diagram of the results proved in this paper. The remaining sections in this paper are organized as follows. In Section 4 we recall useful notions and definitions. In Section 5 we define oracle reductions, and prove how to create IOP protocols from RS oracle reductions and RS proximity tests. In the full version, we define and construct trace embeddings, describe our succinct lincheck protocol, describe an oracle reduction from R1CS automata to testing proximity to the Reed–Solomon code, describe an oracle reduction from R1CS machines to testing proximity to the Reed–Solomon codem, and finally prove Theorems 1 and 2.

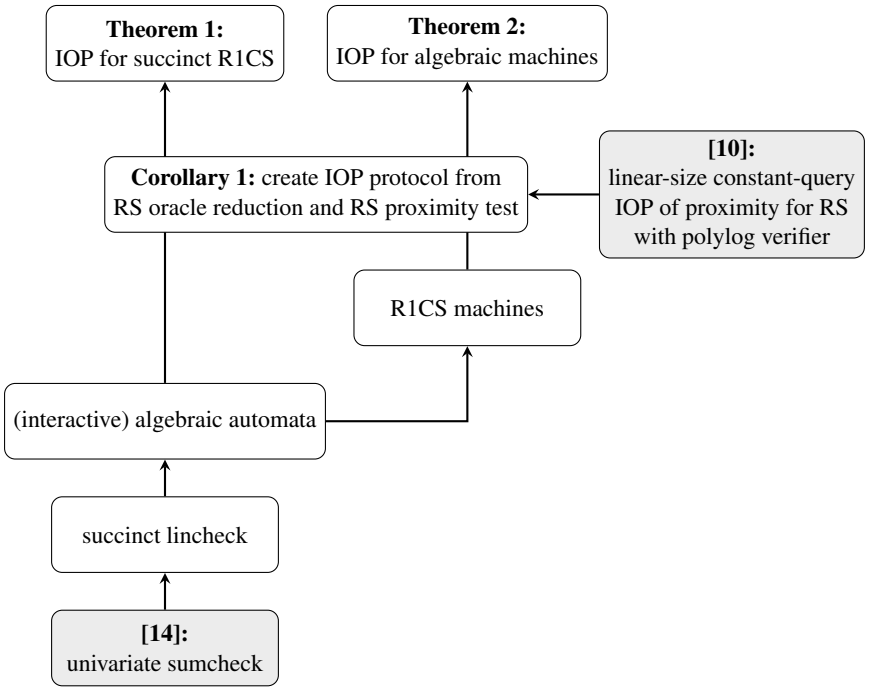

Fig. 1: Diagram of the results in this paper.

4 Preliminaries

Given a relation $\mathcal{R} \subseteq S \times T$, we denote by $\mathcal{L}(\mathcal{R}) \subseteq S$ the set of $s \in S$ such that there exists $t \in T$ with $(s,t) \in \mathcal{R}$; for $s \in S$, we denote by $\mathcal{R}|_s \subseteq T$ the set $\{t \in T : (s,t) \in \mathcal{R}\}$. Given a set S and strings $v, w \in S^n$ for some $n \in \mathbb{N}$, the *fractional Hamming distance* $\Delta(v,w) \in [0,1]$ is $\Delta(v,w) \coloneqq \frac{1}{n}|\{i : v_i \neq w_i\}|$. We denote the concatenation

of two vectors u_1, u_2 by $u_1 \| u_2$, and the concatenation of two matrices A, B by $[A|B]$. All fields $\mathbb{F}$ in this paper are finite, and we denote the finite field of size q by $\mathbb{F}_q$. We say that H is a *subgroup* in $\mathbb{F}$ if it is either a subgroup of $(\mathbb{F}, +)$ (an additive subgroup) or of $(\mathbb{F} \setminus \{0\}, \times)$ (a multiplicative subgroup); we say that H is a *coset* in $\mathbb{F}$ if it is a coset of a subgroup in $\mathbb{F}$ (possibly the subgroup itself).

4.1 Codes and polynomials

The Reed–Solomon code. Given a subset S of a field $\mathbb{F}$ and *rate* $\rho \in (0, 1]$, we denote by $\mathrm{RS}[S, \rho] \subseteq \mathbb{F}^S$ all evaluations over S of univariate polynomials of degree less than $\rho|S|$. Namely, a word $c \in \mathbb{F}^S$ is in $\mathrm{RS}[S, \rho]$ if there is a polynomial of degree less than $\rho|S|$ that, for every $a \in S$, evaluates to c_a at a. We denote by $\mathrm{RS}[S, (\rho_1, \ldots, \rho_n)] \coloneqq \prod_{i=1}^{n} \mathrm{RS}[S, \rho_i]$ the interleaving of Reed–Solomon codes with rates $\rho_1, \ldots, \rho_n$.

Representations of polynomials. We frequently move from univariate polynomials over $\mathbb{F}$ to their evaluations on subsets of $\mathbb{F}$, and back. We use plain letters like f, g, h, π to denote *evaluations* of polynomials, and "hatted letters" $\hat{f}, \hat{g}, \hat{h}, \hat{\pi}$ to denote corresponding polynomials. This bijection is well-defined only if the size of the evaluation domain is larger than the degree. If $f \in \mathrm{RS}[S, \rho]$ for $S \subseteq \mathbb{F}$ and $\rho \in (0, 1]$, then $\hat{f}$ is the unique polynomial of degree less than $\rho|S|$ whose evaluation on S equals f. Likewise, if $\hat{f} \in \mathbb{F}[X]$ with $\deg(\hat{f}) < \rho|S|$, then $f_S \coloneqq \hat{f}|_S \in \mathrm{RS}[S, \rho]$ (but we drop the subscript when the subset is clear from context).

Vanishing polynomials. Let $\mathbb{F}$ be a finite field, and $S \subseteq \mathbb{F}$. We denote by $\mathbb{Z}_S$ the unique non-zero monic polynomial of degree at most $|S|$ that is zero everywhere on S; $\mathbb{Z}_S$ is called the *vanishing polynomial* of S. In this work we use efficiency properties of vanishing polynomials for sets S that have group structure.

If S is a multiplicative subgroup of $\mathbb{F}$, then $\mathbb{Z}_S(X) = X^{|S|} - 1$, and so $\mathbb{Z}_S(X)$ can be evaluated at any $\alpha \in \mathbb{F}$ in $O(\log |S|)$ field operations. More generally, if S is a γ-coset of a multiplicative subgroup S_0 (namely, $S = \gamma S_0$) then $\mathbb{Z}_S(X) = \gamma^{|S|}\mathbb{Z}_{S_0}(X/\gamma) = X^{|S|} - \gamma^{|S|}$.

If S is an (affine) subspace of $\mathbb{F}$, then $\mathbb{Z}_S$ is called an *(affine) subspace polynomial*. In this case, there exist coefficients $c_0, \ldots, c_k \in \mathbb{F}$, where $k \coloneqq \dim(S)$, such that $\mathbb{Z}_S(X) = X^{p^k} + \sum_{i=1}^{k} c_i X^{p^{i-1}} + c_0$ (if S is linear then $c_0 = 0$). Hence, $\mathbb{Z}_S(X)$ can be evaluated at any $\alpha \in \mathbb{F}$ in $O(k \log p) = O(\log |S|)$ operations. Such polynomials are called *linearized* because they are $\mathbb{F}_p$-affine maps: if $S = S_0 + \gamma$ for a subspace $S_0 \subseteq \mathbb{F}$ and shift $\gamma \in \mathbb{F}$, then $\mathbb{Z}_S(X) = \mathbb{Z}_{S_0}(X - \gamma) = \mathbb{Z}_{S_0}(X) - \mathbb{Z}_{S_0}(\gamma)$, and $\mathbb{Z}_{S_0}$ is an $\mathbb{F}_p$-linear map. The coefficients $c_0, \ldots, c_k$ can be derived from a description of S (any basis of S_0 and the shift γ) in $O(k^2 \log p)$ field operations (see [35, Chapter 3.4] and [12, Remark C.8]).

Lagrange polynomials. For $\mathbb{F}$ a finite field, $S \subseteq \mathbb{F}$, $a \in S$, we denote by $L_{S,a}$ the unique polynomial of degree less than $|S|$ such that $L_{S,a}(a) = 1$ and $L_{S,a}(b) = 0$ for all $b \in S \setminus \{a\}$. Note that

$$L_{S,a}(X) = \frac{\prod_{b \in S \setminus \{a\}} (X - b)}{\prod_{b \in S \setminus \{a\}} (a - b)} = \frac{L'_S(X)}{L'_S(a)} \ ,$$

where $L'_S(X)$ is the polynomial $\mathbb{Z}_S(X)/(X-a)$. For additive and multiplicative subgroups S and $a \in S$, we can evaluate $L_{S,a}(X)$ at any $\alpha \in \mathbb{F}$ in $\mathrm{polylog}(|S|)$ field operations. This is because an arithmetic circuit for L'_S can be efficiently derived from an arithmetic circuit for $\mathbb{Z}_S$ [41].

4.2 Interactive oracle proofs

The information-theoretic protocols in this paper are *Interactive Oracle Proofs* (IOPs) [15, 40], which combine aspects of Interactive Proofs [4, 28] and Probabilistically Checkable Proofs [5, 3, 2], and also generalize the notion of Interactive PCPs [33].

A k-round public-coin IOP has k rounds of interaction. In the i-th round of interaction, the verifier sends a uniformly random message m_i to the prover; then the prover replies with a message π_i to the verifier. After k rounds of interaction, the verifier makes some queries to the oracles it received and either accepts or rejects.

An *IOP system* for a relation $\mathcal{R}$ with round complexity k and soundness error ε is a pair (P, V), where P, V are probabilistic algorithms, that satisfies the following properties. (See [15, 40] for details.)

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(P(\mathtt{x}, \mathtt{w}), V(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability 1.

Soundness: For every instance $\mathtt{x} \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{P}$, $(\tilde{P}, V(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability at most $\varepsilon(n)$.

Like the IP model, a fundamental measure of efficiency is the round complexity k. Like the PCP model, two additional fundamental measures of efficiency are the *proof length* p, which is the total number of alphabet symbols in all of the prover's messages, and the *query complexity* q, which is the total number of locations queried by the verifier across all of the prover's messages.

We say that an IOP system is *non-adaptive* if the verifier queries are non-adaptive, namely, the queried locations depend only on the verifier's inputs and its randomness. All of our IOP systems will be non-adaptive.

IOPs of proximity An IOP *of Proximity* extends an IOP the same way that PCPs of Proximity extend PCPs. An *IOPP system* for a relation $\mathcal{R}$ with round complexity k, soundness error ε, and proximity parameter δ is a pair (P, V) that satisfies the following properties.

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(P(\mathtt{x}, \mathtt{w}), V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability 1.

Soundness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ with $\Delta(\mathtt{w}, \mathcal{R}|_{\mathtt{x}}) \geq \delta(n)$ and unbounded malicious prover $\tilde{P}$, $(\tilde{P}, V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability at most $\varepsilon(n)$.

Efficiency measures for IOPPs are as for IOPs, except that we also count queries to the witness: if V makes at most $\mathsf{q}_{\mathtt{w}}$ queries to $\mathtt{w}$ and at most q_π queries to prover messages, the query complexity is $\mathsf{q} := \mathsf{q}_{\mathtt{w}} + \mathsf{q}_\pi$.

5 Oracle reductions

We define *interactive oracle reductions* (henceforth just *oracle reductions*), which, informally, are reductions from computational problems to the problem of testing membership of collections of oracles in a code.

The main result in this section is Lemma 4 (and an implication of it, Corollary 1), which enables the construction of IOPs by modularly combining oracle reductions and proximity tests. The ideas underlying oracle reductions are not new. Essentially all known constructions of PCPs/IPCPs/IOPs consist of two parts: (1) an encoding, typically via an algebraic code, that endows the witness with robust structure (often known as *arithmetization*); and (2) a procedure that locally tests this encoding (often known as *low-degree testing*).

Oracle reductions provide a formal method of constructing proof systems according to this framework. We use them to express results in the full version of the paper, which significantly simplifies exposition. Additionally, expressing our results as oracle reductions enables us to consider the efficiency of the oracle reduction itself as a separate goal from the efficiency of the low-degree test. In particular, future improvements in low-degree testing will lead immediately to improvements in our protocols.

This section has two parts: in Section 5.1 we define oracle reductions; then in Section 5.2, we introduce a special case of oracle reductions where the target code is the Reed–Solomon (RS) code. For this special case we give additional lemmas: we show that it suffices to prove a weaker soundness property, because it generically implies standard soundness; also, we show that all such oracle reductions admit a useful optimization which reduces the number of low-degree tests needed to a single one.

5.1 Definitions

Informally, an oracle reduction is an interactive public-coin protocol between a prover and a verifier that reduces membership in a language to a promise problem on oracles sent by the prover during the protocol.

In more detail, an oracle reduction from a language $\mathcal{L} \subseteq X$ to a relation $\mathcal{R}' \subseteq X' \times \Sigma^s$ is an interactive protocol between a prover and a verifier that both receive an instance $\mathbb{x} \in X$, where in each round the verifier sends a message and the prover replies with an oracle (or several oracles), as in the IOP model. Unlike in an IOP, the verifier *does not make any queries*. Instead, after the interaction the verifier outputs a list of claims of the form "$(\mathbb{x}, \mathbf{\Pi}) \in \mathcal{R}'$", which may depend on the verifier's randomness, where $\mathbb{x}' \in X'$ and $\mathbf{\Pi}$ is a deterministic oracle algorithm that specifies a string in Σ^s as follows: the i-th entry in Σ^s is computed as $\mathbf{\Pi}^{\pi_1,\ldots,\pi_r}(i)$, where π_j is the oracle sent by the prover in the j-th round. The reduction has the property that if $\mathbb{x} \in \mathcal{L}$ then all claims output by the verifier are true, and if instead $\mathbb{x} \notin \mathcal{L}$ then (with high probability over the verifier's randomness) at least one claim is false.

We refer to each oracle algorithm $\mathbf{\Pi}^{(j)}$ as a *virtual oracle* because $\mathbf{\Pi}^{(j)}$ represents an oracle that is derived from oracles sent by the prover. We are interested in virtual oracles $\mathbf{\Pi}^{(j)}$ where, for each i, the number of queries $\mathbf{\Pi}^{\pi_1,\ldots,\pi_r}(i)$ makes to the oracles is small. For simplicity, we also assume that the algorithms are non-adaptive in that the queried locations are independent of the answers to the queries.

A crucial property is that virtual oracles with small locality compose well, which allows us to compose oracle reductions. For this we need an oracle reduction *of proximity* (Definition 5), which we can view as an oracle reduction from a relation $\mathcal{R} \subseteq X \times \Sigma^s$ to another relation $\mathcal{R}' \subseteq X' \times \Sigma^{s'}$. Then we can construct an oracle reduction from $\mathcal{L}$ to $\mathcal{R}'$ by composing an oracle reduction A from $\mathcal{L}$ to $\mathcal{R}'$ with an oracle reduction of proximity B from $\mathcal{R}'$ to $\mathcal{R}''$. Such a reduction may output virtual oracles of the form $\mathbf{\Pi}_B^{\mathbf{\Pi}_A}$ where $\mathbf{\Pi}_B$ is a virtual oracle output by B and $\mathbf{\Pi}_A$ is a virtual oracle output by A. This can be expressed as a standard virtual oracle with access to the prover messages, and if $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$ have small locality then so does $\mathbf{\Pi}_B^{\mathbf{\Pi}_A}$.

We now formalize the foregoing discussion, starting with the notion of a virtual oracle. Since the virtual oracles in this work are non-adaptive, we specify them via query ("pre-processing") and answer ("post-processing") algorithms. The query algorithm receives an index $i \in [s]$ and computes a list of locations to be queried across oracles. The answer algorithm receives the same index i, and answers to the queries, and computes the value of the virtual oracle at location i. In other words, the answer algorithm computes the value of the virtual oracle at the desired location from the values of the real oracles at the queried locations.

Definition 3. *A* **virtual oracle** $\mathbf{\Pi}$ *of length s over an alphabet Σ is a pair of deterministic polynomial-time algorithms (Q, A). Given any oracles $\pi_1, \ldots, \pi_r$ of appropriate sizes, these algorithms define an oracle $\mathbf{\Pi} \in \Sigma^s$ given by $\mathbf{\Pi}[\pi_1, \ldots, \pi_r](i) \coloneqq A(i, (\pi_j[k])_{(j,k)\in Q(i)})$ for $i \in [s]$.* $\mathbf{\Pi}$ *is* ℓ**-local** *if* $\max_{i\in[s]} |Q(i)| \leq \ell$.

Observe that the definition of a virtual oracle given above is equivalent to saying that $\mathbf{\Pi}$ is an algorithm with non-adaptive query access to $\pi_1, \ldots, \pi_r$. Where convenient we will use this perspective.

We now define the notion of an oracle reduction. Since in this work we primarily deal with relations, rather than languages, we define our reductions accordingly.

Definition 4. *An* **oracle reduction** *from a relation $\mathcal{R}$ to a relation $\mathcal{R}'$ with base alphabet Σ is an interactive protocol between a prover P and verifier V that works as follows. The prover P takes as input an instance-witness pair $(\mathbb{x}, \mathbb{w})$ and the verifier V takes as input the instance $\mathbb{x}$. In each round, V sends a message $m_i \in \{0,1\}^*$, and P replies with an oracle $\pi_i \in \Sigma_i^*$ over an alphabet $\Sigma_i = \Sigma^{s_i}$; let $\pi_1, \ldots, \pi_r$ be all oracles sent.*[5] *After the interaction, V outputs a list of instances $(\mathbb{x}^{(1)}, \ldots, \mathbb{x}^{(m)})$ and a list of virtual oracles $(\mathbf{\Pi}^{(1)}, \ldots, \mathbf{\Pi}^{(m)})$ over alphabets $\Sigma_1', \ldots, \Sigma_m'$ respectively, where $\Sigma_i' = \Sigma^{s_i'}$.*

We say that the oracle reduction has **soundness error** ϵ *and* **distance** δ *if the following conditions hold.*

- Completeness: *If $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ then, with probability 1 over the verifier's randomness, for every $j \in [m]$ it holds that $\left(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r]\right) \in \mathcal{R}'$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*
- Soundness:[6] *If $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ then for any prover $\tilde{P}$, with probability $1 - \epsilon$ over the verifier's randomness, there exists $j \in [m]$ such that $\Delta(\mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r], \mathcal{R}'|_{\mathbb{x}^{(j)}}) > \delta$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*

[5] Sometimes it is convenient to allow the prover to reply with multiple oracles $\pi_{i,1}, \pi_{i,2}, \ldots$; all discussions extend to this case.

[6] This is analogous to the "interactive soundness error" ε_{i} in [14].

An oracle reduction is *public coin* if all of the verifier's messages consist of uniform randomness. All of the oracle reductions we present in this paper are public coin. Note that we can always choose the base alphabet Σ to be $\{0,1\}$, but it will be convenient for us to use a larger base alphabet.

This above definition can be viewed as extending the notion of *linear-algebraic CSPs* [11], and indeed Lemma 4 below gives a construction similar to the "canonical" PCP described in that work.

It will be useful to *compose* oracle reductions. As in the PCP setting, for this we will require an object with a stronger *proximity soundness* property.

Definition 5. *An* **oracle reduction of proximity** *is as in Definition 4 except that, for a given proximity parameter $\delta_0 \in (0,1)$, the soundness condition is replaced by the following one.*

- Proximity soundness: *If $(\mathbb{x}, \mathbb{w})$ is such that $\Delta(\mathbb{w}, \mathcal{R}|_{\mathbb{x}}) > \delta_0$ then for any prover $\tilde{P}$, with probability $1-\epsilon$ over the verifier's randomness, there exists $j \in [m]$ such that $\Delta(\mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r], \mathcal{R}'|_{\mathbb{x}^{(j)}}) > \delta(\delta_0)$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*

In the PCPP literature the foregoing soundness property is usually known as *robust soundness*, and the condition is expressed in terms of expected distance. The definition given here is more convenient for us.

Efficiency measures. There are several efficiency measures that we study for an oracle reduction.

- An oracle reduction has r *rounds* if the interactive protocol realizing it has r rounds.
- An oracle reduction has m *virtual oracles* and *locality* ℓ if the verifier outputs at most m virtual oracles $\{\mathbf{\Pi}^{(j)} = (Q_j, A_j)\}_{j\in[m]}$, and it holds that $\max_{i\in[s]} |\cup_{j=1}^{m} Q_j(i)| \leq \ell$. Note that the answer to a single query may consist of multiple symbols over the base alphabet Σ, but we count the query only once.
- An oracle reduction has *length* $s = \sum_{i=1}^{r} s_i|\pi_i|$ over the base alphabet. Its length in *bits* is $s \log |\Sigma|$.

Other efficiency measures include the running time of the prover and of the verifier.

Oracle reductions combine naturally with proofs of proximity to produce IOPs. The following lemma is straightforward, and we state it without proof.

Lemma 4. *Suppose that there exist:*

(i) an r-round oracle reduction from $\mathcal{R}$ to $\mathcal{R}'$ over base alphabet Σ with soundness error ϵ, distance δ, length s, locality ℓ, and m virtual oracles;

(ii) an r'-round IOPP for $\mathcal{R}'$ over alphabet Σ with soundness error ϵ', proximity parameter $\delta' \leq \delta$, length s', and query complexity $(\mathsf{q}_{\mathbb{w}}, \mathsf{q}_{\pi})$.

Then there exists an $(r + mr')$-round IOP for $\mathcal{R}$ with soundness error $\epsilon + \epsilon'$, length $s + s' \cdot m$ over Σ, and query complexity $(\mathsf{q}_{\mathbb{w}} \cdot \ell + \mathsf{q}_{\pi}) \cdot m$.

5.2 Reed–Solomon oracle reductions

In this work we focus on a special class of oracle reductions, in which we reduce to membership in the Reed–Solomon code, and where the virtual oracles have a special

form. These reductions coincide with "RS-encoded IOPs" [14, Definition 4.6], which we recast in the language of virtual oracles.

We first define the notion of a *rational constraint*, a special type of virtual oracle that is "compatible" with the (interleaved) Reed–Solomon code.

Definition 6. *A* **rational constraint** *is a virtual oracle* $\mathbf{\Pi} = (Q, A)$ *over a finite field* $\mathbb{F}$ *where* $Q(\alpha) = ((1, \alpha), \ldots, (r, \alpha))$ *and* $A(\alpha, \beta_1, \ldots, \beta_r) = N(\alpha, \beta_1, \ldots, \beta_r)/D(\alpha)$*, for two arithmetic circuits (without division gates)* $N \colon \mathbb{F}^{\sum_i s_i} \to \mathbb{F}$ *and* $D \colon \mathbb{F} \to \mathbb{F}$*.*

A Reed–Solomon (RS) oracle reduction is a reduction from some relation to membership in the Reed–Solomon code, where additionally every oracle is a rational constraint.

Definition 7. *A* **Reed–Solomon (RS) oracle reduction** *over a domain* $L \subseteq \mathbb{F}$ *is an oracle reduction, over the base alphabet* $\mathbb{F}$*, from a relation* $\mathcal{R}$ *to the interleaved Reed–Solomon relation*

$$\mathcal{R}^*_{\mathrm{RS}} \coloneqq \left\{ (\boldsymbol{\rho}, f) \text{ s.t. } \boldsymbol{\rho} \in (0,1]^*,\ f \colon L \to \mathbb{F} \text{ is a codeword in } \mathrm{RS}\left[L, \boldsymbol{\rho}\right] \right\}$$

where every virtual oracle output by the verifier is a rational constraint, except for a special instance $(\boldsymbol{\rho}_0, \mathbf{\Pi}_0)$*, which the verifier must output.* $\mathbf{\Pi}_0$*, over alphabet* $\mathbb{F}^{\sum_i s_i}$*, is given by* $\mathbf{\Pi}_0(\alpha) = (\pi_1(\alpha), \ldots, \pi_r(\alpha))$ *(i.e., it is a stacking of the oracles sent by the prover).*

In this work we will assume throughout that L comes a family of subgroups (of a family of fields) such that there is an encoding algorithm for the Reed–Solomon code on domain L

Note that $\mathbf{\Pi}_0$ is not a rational constraint because its alphabet is not $\mathbb{F}$. Later we will also refer to the *non-interleaved* Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}} \coloneqq \{(\rho, f) : \rho \in (0,1], f \in \mathrm{RS}\left[L, \rho\right]\}$.

RS oracle reductions have a useful property: if the soundness condition holds for $\delta = 0$, then the soundness condition also holds for a distance $\delta > 0$ related to the *maximum rate* of the reduction. Informally, the maximum rate is the maximum over the (prescribed) rates of codewords sent by the prover and those induced by the verifier's rational constraints. To define it, we need notation for the *degree* and *rate* of a circuit.

Definition 8. *The* **degree** *of an arithmetic circuit* $C \colon \mathbb{F}^{1+\ell} \to \mathbb{F}$ *on input degrees* $d_1, \ldots, d_\ell \in \mathbb{N}$*, denoted* $\deg(C; d_1, \ldots, d_\ell)$*, is the smallest integer* e *such that for all* $p_i \in \mathbb{F}^{\leq d_i}[X]$ *there exists a polynomial* $q \in \mathbb{F}^{\leq e}[X]$ *such that* $C(X, p_1(X), \ldots, p_\ell(X)) \equiv q(X)$*. Given domain* $L \subseteq \mathbb{F}$ *and rates* $\boldsymbol{\rho} \in (0,1]^\ell$*, the* **rate** *of* C *is* $\mathrm{rate}(C; \boldsymbol{\rho}) \coloneqq \deg(C; \rho_1|L|, \ldots, \rho_\ell|L|)/|L|$*. (The domain* L *will be clear from context.) Note that if* $\ell = 0$ *then this notion of degree coincides with the usual one (namely,* $\deg(C)$ *is the degree of the polynomial described by* C*), and* $\mathrm{rate}(C) \coloneqq \deg(C)/|L|$*.*

An oracle reduction has ***maximum rate*** ρ^* if, for every rational constraint $(\sigma, \mathbf{\Pi})$ output by the verifier, $\max(\mathrm{rate}(N; \boldsymbol{\rho}_0), \sigma + \mathrm{rate}(D)) \leq \rho^*$. This expression is motivated by the proof of the following lemma; see [14, Proof of Theorem 9.1] for details.

Lemma 5. *Suppose that an RS oracle reduction with maximum rate ρ^* satisfies the following* **weak soundness** *condition: if $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ then for any prover $\tilde{P}$, with probability $1-\epsilon$ over the verifier's randomness, there exists $j \in [m]$ such that $(\rho^{(j)}, \mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_n]) \notin \mathcal{R}_{\mathrm{RS}}$. Then the reduction satisfies the standard soundness condition (see Definition 4) with soundness error ϵ and distance $\delta \coloneqq \frac{1}{2}(1-\rho^*)$.*

This means that for the oracle reductions in this paper *we need only establish weak soundness*. Also, one can see that RS oracle reductions have locality r (the number of rounds), since $|Q(\alpha)| = r$ for all $\alpha \in L$.

The following lemma shows that, for RS oracle reductions, it suffices to run the proximity test on a single virtual oracle. This reduces the query complexity and proof length when we apply Lemma 4.

Lemma 6. *Suppose that there exists an r-round RS oracle reduction from $\mathcal{R}$ over domain L, m virtual oracles, soundness error ϵ, maximum rate ρ^*, and distance δ. Then there is an r-round oracle reduction from $\mathcal{R}$ to the* non-interleaved *Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}}$ with locality r,* **one** *virtual oracle, soundness error $\epsilon + |L|/|\mathbb{F}|$, maximum rate ρ^*, and distance $\min(\delta, (1-\rho^*)/3, (1-2\rho^*)/2)$.*

Proof. Implicit in [14, Proof of Theorem 9.1], where it follows from [19].

Combining Lemmas 4 to 6 yields the following useful corollary. We invoke it, in the full version, on the two main building blocks obtained in this paper in order to prove our main result.

Corollary 1. *Suppose that there exist:*

(i) an r-round RS oracle reduction from $\mathcal{R}$ over domain L, m virtual oracles, length s and rate ρ^* that satisfies the weak soundness condition with soundness error ϵ;

(ii) an r'-round IOP of proximity for $\mathcal{R}_{\mathrm{RS}}$ with soundness error ϵ', proximity parameter $\delta' < \min((1-\rho^*)/3, (1-2\rho^*)/2)$, length p and query complexity $(\mathsf{q}_{\mathsf{w}}, \mathsf{q}_{\pi})$.

Then there exists an $(r + r')$-round IOP for $\mathcal{R}$ with soundness error $\epsilon + \epsilon' + \frac{|L|}{|\mathbb{F}|}$, length $s + \mathsf{p}$ and query complexity $\mathsf{q}_{\mathsf{w}} \cdot r + \mathsf{q}_{\pi}$.

Acknowledgments

We thank Michael Forbes for helpful discussions. This work was supported in part by: donations from the Ethereum Foundation and the Interchain Foundation.

References

1. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple construction of almost k-wise independent random variables. Random Structures and Algorithms **3**(3), 289–304 (1992)
2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. Journal of the ACM **45**(3), 501–555 (1998), preliminary version in FOCS '92.
3. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. Journal of the ACM **45**(1), 70–122 (1998), preliminary version in FOCS '92.

4. Babai, L.: Trading group theory for randomness. In: Proceedings of the 17th Annual ACM Symposium on Theory of Computing. pp. 421–429. STOC '85 (1985)
5. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing. pp. 21–32. STOC '91 (1991)
6. Ben-Sasson, E., Bentov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., Virza, M.: Computational integrity with a public random string from quasi-linear pcps. In: Proceedings of the 36th Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 551–579. EUROCRYPT '17 (2017)
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018)
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed–Solomon interactive oracle proofs of proximity. In: Proceedings of the 45th International Colloquium on Automata, Languages and Programming. pp. 14:1–14:17. ICALP '18 (2018)
9. Ben-Sasson, E., Chiesa, A., Forbes, M.A., Gabizon, A., Riabzev, M., Spooner, N.: Zero knowledge protocols from succinct constraint detection. In: Proceedings of the 15th Theory of Cryptography Conference. pp. 172–206. TCC '17 (2017)
10. Ben-Sasson, E., Chiesa, A., Gabizon, A., Riabzev, M., Spooner, N.: Interactive oracle proofs with constant rate and query complexity. In: Proceedings of the 44th International Colloquium on Automata, Languages and Programming. pp. 40:1–40:15. ICALP '17 (2017)
11. Ben-Sasson, E., Chiesa, A., Gabizon, A., Virza, M.: Quasilinear-size zero knowledge from linear-algebraic PCPs. In: Proceedings of the 13th Theory of Cryptography Conference. pp. 33–64. TCC '16-A (2016)
12. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In: Proceedings of the 4th Innovations in Theoretical Computer Science Conference. pp. 401–414. ITCS '13 (2013)
13. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: Proceedings of the 45th ACM Symposium on the Theory of Computing. pp. 585–594. STOC '13 (2013)
14. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 103–128. EUROCRYPT '19 (2019), full version available at `https://eprint.iacr.org/2018/828`
15. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Proceedings of the 14th Theory of Cryptography Conference. pp. 31–60. TCC '16-B (2016)
16. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.: Short PCPs verifiable in polylogarithmic time. In: Proceedings of the 20th Annual IEEE Conference on Computational Complexity. pp. 120–134. CCC '05 (2005)
17. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust PCPs of proximity, shorter PCPs, and applications to coding. SIAM Journal on Computing **36**(4), 889–974 (2006)
18. Ben-Sasson, E., Kaplan, Y., Kopparty, S., Meir, O., Stichtenoth, H.: Constant rate PCPs for Circuit-SAT with sublinear query complexity. In: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science. pp. 320–329. FOCS '13 (2013)
19. Ben-Sasson, E., Kopparty, S., Saraf, S.: Worst-case to average case reductions for the distance to a code. In: Proceedings of the 33rd ACM Conference on Computer and Communications Security. pp. 24:1–24:23. CCS '18 (2018)
20. Ben-Sasson, E., Sudan, M.: Short PCPs with polylog query complexity. SIAM Journal on Computing **38**(2), 551–607 (2008), preliminary version appeared in STOC '05.

21. Ben-Sasson, E., Sudan, M., Vadhan, S., Wigderson, A.: Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing. pp. 612–621. STOC '03 (2003)
22. Blum, M., Kannan, S.: Designing programs that check their work. Journal of the ACM **42**(1), 269–291 (1995), preliminary version in STOC '89.
23. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security. pp. 336–365 (2017)
24. Bowe, S., Gurkan, K., Tromer, E., B?nz, B., Chalkias, K., Genkin, D., Grigg, J., Hopwood, D., Law, J., Poelstra, A., abhi shelat, Venkitasubramaniam, M., Virza, M., Wahby, R.S., Wuille, P.: Implementation track proceeding. Tech. rep., ZKProof Standards (2018), `https://zkproof.org/documents.html`
25. Dinur, I.: The PCP theorem by gap amplification. Journal of the ACM **54**(3), 12 (2007)
26. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. Journal of the ACM **43**(2), 268–292 (1996), preliminary version in FOCS '91.
27. Goldreich, O., Sudan, M.: Locally testable codes and PCPs of almost-linear length. Journal of the ACM **53**, 558–655 (July 2006), preliminary version in STOC '02.
28. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989), preliminary version appeared in STOC '85.
29. Gurevich, Y., Shelah, S.: Nearly linear time. In: Logic at Botik '89, Symposium on Logical Foundations of Computer Science. pp. 108–118 (1989)
30. Guruswami, V., Indyk, P.: Linear-time encodable/decodable codes with near-optimal rate. IEEE Transactions on Information Theory **51**(10), 3393–3400 (2005), preliminary version appeared in STOC '03.
31. Harsha, P., Sudan, M.: Small PCPs with low query complexity. Computational Complexity **9**(3–4), 157–201 (Dec 2000), preliminary version in STACS '01.
32. Ishai, Y., Mahmoody, M., Sahai, A., Xiao, D.: On zero-knowledge PCPs: Limitations, simplifications, and applications (2015), available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`
33. Kalai, Y., Raz, R.: Interactive PCP. In: Proceedings of the 35th International Colloquium on Automata, Languages and Programming. pp. 536–547. ICALP '08 (2008)
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing. pp. 723–732. STOC '92 (1992)
35. Lidl, R., Niederreiter, H.: Finite Fields. Cambridge University Press, second edition edn. (1997)
36. Lipton, R.J.: New directions in testing. In: Proceedings of a DIMACS Workshop in Distributed Computing And Cryptography. pp. 191–202 (1989)
37. Micali, S.: Computationally sound proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000), preliminary version appeared in FOCS '94.
38. Mie, T.: Short PCPPs verifiable in polylogarithmic time with o(1) queries. Annals of Mathematics and Artificial Intelligence **56**, 313–338 (2009)
39. Polishchuk, A., Spielman, D.A.: Nearly-linear size holographic proofs. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing. pp. 194–203. STOC '94 (1994)
40. Reingold, O., Rothblum, R., Rothblum, G.: Constant-round interactive proofs for delegating computation. In: Proceedings of the 48th ACM Symposium on the Theory of Computing. pp. 49–62. STOC '16 (2016)
41. Shpilka, A., Yehudayoff, A.: Arithmetic circuits: A survey of recent results and open questions. Foundations and Trends in Theoretical Computer Science **5**(3-4), 207–388 (2010)

42. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. IEEE Transactions on Information Theory **42**(6), 1723–1731 (1996), preliminary version appeared in STOC '95.