Zero Knowledge Protocols from Succinct Constraint Detection

Eli Ben-Sasson
eli@cs.technion.ac.il
Technion

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Michael A. Forbes
miforbes@csail.mit.edu
Stanford University

Ariel Gabizon
ariel@z.cash
ZcashCo*

Michael Riabzev
mriabzev@cs.technion.ac.il
Technion

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

September 20, 2017

Abstract

We study the problem of constructing proof systems that achieve both soundness and zero knowledge unconditionally (without relying on intractability assumptions). Known techniques for this goal are primarily *combinatorial*, despite the fact that constructions of interactive proofs (IPs) and probabilistically checkable proofs (PCPs) heavily rely on *algebraic* techniques to achieve their properties.

We present simple and natural modifications of well-known 'algebraic' IP and PCP protocols that achieve unconditional (perfect) zero knowledge in recently introduced models, overcoming limitations of known techniques.

- We modify the PCP of Ben-Sasson and Sudan [BS08] to obtain zero knowledge for $\mathbf{NEXP}$ in the model of Interactive Oracle Proofs [BCS16, RRR16], where the verifier, in each round, receives a PCP from the prover.
- We modify the IP of Lund, Fortnow, Karloff, and Nisan [LFKN92] to obtain zero knowledge for $\#\mathbf{P}$ in the model of Interactive PCPs [KR08], where the verifier first receives a PCP from the prover and then interacts with him.

The simulators in our zero knowledge protocols rely on solving a problem that lies at the intersection of coding theory, linear algebra, and computational complexity, which we call the *succinct constraint detection* problem, and consists of detecting dual constraints with polynomial support size for codes of exponential block length. Our two results rely on solutions to this problem for fundamental classes of linear codes:

- An algorithm to detect constraints for Reed–Muller codes of exponential length. This algorithm exploits the Raz–Shpilka [RS05] deterministic polynomial identity testing algorithm, and shows, to our knowledge, a first connection of algebraic complexity theory with zero knowledge.
- An algorithm to detect constraints for PCPs of Proximity of Reed–Solomon codes [BS08] of exponential degree. This algorithm exploits the recursive structure of the PCPs of Proximity to show that small-support constraints are "locally" spanned by a small number of small-support constraints.

Keywords: probabilistically checkable proofs, interactive proofs, sumcheck, zero knowledge, polynomial identity testing

*Work conducted while at Technion.

Contents

1 Introduction

The study of interactive proofs (IPs) [BM88, GMR89] that unconditionally achieve zero knowledge [GMR89] has led to a rich theory, with connections well beyond zero knowledge. For example, the class of languages with statistical zero knowledge IPs, which we denote by $\mathbf{SZK}$-$\mathbf{IP}$, has complete problems that make no reference to either zero knowledge or interaction [SV03, GV99] and is closed under complement [Oka00, Vad99]. Despite the fact that all $\mathbf{PSPACE}$ languages have IPs [Sha92], $\mathbf{SZK}$-$\mathbf{IP}$ is contained in $\mathbf{AM} \cap \mathbf{coAM}$, and thus $\mathbf{NP}$ is not in $\mathbf{SZK}$-$\mathbf{IP}$ unless the polynomial hierarchy collapses [BHZ87]; one consequence is that Graph Non-Isomorphism is unlikely to be NP-complete. Moreover, constructing $\mathbf{SZK}$-$\mathbf{IP}$ for a language is equivalent to constructing instance-dependent commitments for the language [IOS97, OV08], and has connections to other fundamental information-theoretic notions like randomized encodings [AR16, VV15] and secret-sharing schemes [VV15].

Unconditional zero knowledge in other models behaves very differently. Ben-Or, Goldwasser, Kilian, and Wigderson [BGKW88] introduced the model of multi-prover interactive proofs (MIPs) and showed that *all* such proofs can be made zero knowledge unconditionally. The analogous statement for IPs is equivalent to the existence of one-way functions, as shown by [GMR89, IY87, BGG$^+$88] in one direction and by [Ost91, OW93] in the other (unless $\mathbf{BPP} = \mathbf{PSPACE}$, in which case the statement is trivial). Subsequent works not only established that all $\mathbf{NEXP}$ languages have MIPs [BFL91], but also led to formulating probabilistically checkable proofs (PCPs) and proving the celebrated PCP Theorem [FRS88, BFLS91, FGL$^+$96, AS98, ALM$^+$98], as well as constructing statistical zero knowledge PCPs [KPT97] and applying them to black-box cryptography [IMS12, IMSX15].

The theory of zero knowledge for these types of proofs, however, is not as rich as in the case of IPs. Most notably, known techniques to achieve zero knowledge MIPs or PCPs are limited, and come with caveats. Zero knowledge MIPs are obtained via complex generic transformations [BGKW88], assume the full power of the PCP Theorem [DFK$^+$92], or support only languages in $\mathbf{NP}$ [LS95]. Zero knowledge PCPs are obtained via a construction that incurs polynomial blowups in proof length and requires the honest verifier to adaptively query the PCP [KPT97]. Alternative approaches are not known, despite attempts to find them. For example, [IWY16] apply PCPs to leakage-resilient circuits, obtaining PCPs for $\mathbf{NP}$ that do have a non-adaptive honest verifier but are only witness indistinguishable.

Even basic questions such as "are there zero-knowledge PCPs of quasilinear-size?" or "are there zero-knowledge PCPs with non-adaptive honest verifiers?" have remained frustratingly hard to answer, despite the fact the answers to these questions are well understood when removing the requirement of zero knowledge. This state of affairs begs the question of whether a richer theory about zero knowledge MIPs and PCPs could be established.

The current situation is that known techniques to achieve zero knowledge MIPs and PCPs are combinatorial, namely they make black-box use of an underlying MIP or PCP, despite the fact that most MIP and PCP constructions have a rich algebraic structure arising from the use of error correcting codes based on evaluations of low-degree polynomials. This separation is certainly an attractive feature, and perhaps even unsurprising: while error-correcting codes are designed to help recover information, zero knowledge proofs are designed to hide it.

Yet, a recent work by Ben-Sasson, Chiesa, Gabizon, and Virza [BCGV16] brings together linear error correcting codes and zero knowledge using an algebraic technique that we refer to as 'masking'. The paper introduces a "2-round PCP" for $\mathbf{NP}$ that unconditionally achieves zero knowledge and, nevertheless, has both quasilinear size and a non-adaptive honest verifier. Their work can be viewed not only as partial progress towards some of the open questions above, but also as studying the power of zero knowledge for a natural extension of PCPs ("multi-round PCPs" as discussed below) with its own motivations and applications [BCS16, RRR16, BCG$^+$17].

The motivation of this work is to understand the power of algebraic tools, such as linear error correcting codes, for achieving zero knowledge unconditionally (without relying on intractability assumptions).

1.1 Results

We present new protocols that unconditionally achieve soundness and zero knowledge in recently suggested models that combine features of PCPs and IPs [KR08, BCS16, RRR16]. Our protocols consist of simple and natural modifications to well-known constructions: the PCP of Ben-Sasson and Sudan [BS08] and the IP for polynomial summation of Lund, Fortnow, Karloff, and Nisan [LFKN92]. By leveraging the linear codes used in these constructions, we reduce the problem of achieving zero knowledge to solving exponentially-large instances of a new linear-algebraic problem that we call *constraint detection*, which we believe to be of independent interest. We design efficient algorithms for solving

this problem for notable linear code families, along the way exploiting connections to algebraic complexity theory and local views of linear codes. We now elaborate on the above by discussing each of our results.

1.1.1 Zero knowledge for non-deterministic exponential time

Two recent works [BCS16, RRR16] independently introduce and study the notion of an *interactive oracle proof* (IOP), which can be viewed as a "multi-round PCP". Informally, an IOP is an IP modified so that, whenever the prover sends to the verifier a message, the verifier does not have to read the message in full but may probabilistically query it. Namely, in every round, the verifier sends the prover a message, and the prover replies with a PCP. IOPs enjoy better efficiency compared to PCPs [BCG$^+$17], and have applications to constructing argument systems [BCS16] and IPs [RRR16].

The aforementioned work of [BCGV16] makes a simple modification to the PCP of Ben-Sasson and Sudan [BS08] and obtains a 2-round IOP for $\mathbf{NP}$ that is perfect zero knowledge, and yet has quasilinear size and a non-adaptive honest verifier. Our first result consists of extending this prior work to all languages in $\mathbf{NEXP}$, positively answering an open question raised there. We do so by constructing, for each time T and query bound b, a suitable IOP for $\mathbf{NTIME}(T)$ that is zero knowledge against query bound b; the result for $\mathbf{NEXP}$ follows by setting b to be super-polynomial.

The foregoing notion of zero knowledge for IOPs directly extends that for PCPs, and requires showing the existence of an algorithm that simulates the view of any (malicious and adaptive) verifier interacting with the honest prover and making at most b queries across all oracles; here, 'view' consists of the answers to queries across all oracles.[1]

Theorem 1.1 (informal statement of Thm. 8.1)**.** *For every time bound T and query bound b, the complexity class $\mathbf{NTIME}(T)$ has 2-round Interactive Oracle Proofs that are perfect zero knowledge against b queries, and where the proof length is $\tilde{O}(T + \mathsf{b})$ and the (honest verifier's) query complexity is $\mathrm{polylog}(T + \mathsf{b})$.*

The prior work of [BCGV16] was "stuck" at $\mathbf{NP}$ because their simulator runs in $\mathrm{poly}(T + \mathsf{b})$ time so that T, b must be polynomially-bounded. In contrast, we achieve all of $\mathbf{NEXP}$ by constructing, for essentially the same simple 2-round IOP, a simulator that runs in time $\mathrm{poly}(\tilde{q} + \log T + \log \mathsf{b})$, where $\tilde{q}$ is the *actual* number of queries made by the malicious verifier. This is an *exponential* improvement in simulation efficiency, and we obtain it by conceptualizing and solving a linear-algebraic problem about Reed–Solomon codes, and their proximity proofs, as discussed in Section 1.1.3.

In sum, our theorem gives new tradeoffs compared to [KPT97]'s result, which gives statistical zero knowledge PCPs for $\mathbf{NTIME}(T)$ with proof length $\mathrm{poly}(T, \mathsf{b})$ and an adaptive honest verifier. We obtain perfect zero knowledge for $\mathbf{NTIME}(T)$, with quasilinear proof length and a non-adaptive honest verifier, at the price of "2 rounds of PCPs".

1.1.2 Zero knowledge for counting problems

Kalai and Raz [KR08] introduce and study the notion of *interactive PCPs* (IPCPs), which "sits in between" IPs and IOPs: the prover first sends the verifier a PCP, and then the prover and verifier engage in a standard IP. IPCPs also enjoy better efficiency compared to PCPs or IPs alone [KR08].

We show how a natural and simple modification of the sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [LFKN92] achieves perfect zero knowledge in the IPCP model, even with a non-adaptive honest verifier. By running this protocol on the usual arithmetization of the counting problem associated to 3SAT, we obtain our second result, which is IPCPs for $\#\mathbf{P}$ that are *perfect zero knowledge against unbounded queries*. This means that there exists a polynomial-time algorithm that simulates the view of any (malicious and adaptive) verifier making any polynomial number of queries to the PCP oracle. Here, 'view' consists of answers to oracle queries and the transcript of interaction with the prover. (In particular, this notion of zero knowledge is a 'hybrid' of corresponding notions for PCPs and IPs.)

Theorem 1.2 (informal statement of Thm. 6.2)**.** *The complexity class $\#\mathbf{P}$ has Interactive PCPs that are perfect zero knowledge against unbounded queries. The PCP proof length is exponential, and the communication complexity of the interaction and the (honest verifier's) query complexity are polynomial.*

Our construction relies on a random self-reducibility property of the sumcheck protocol (see Section 2.2.2 for a summary) and its completeness and soundness properties are straightforward to establish. As in our previous result, the

[1] More precisely, while in a zero knowledge IP or MIP one is required to simulate the entire transcript of interaction (with one or multiple provers), in a zero knowledge IOP or PCP one is merely required to simulate answers to the oracle queries but not the entire oracle.

"magic" lies in the construction of the simulator, which must solve the same type of exponentially-large linear-algebraic problem, except that this time it is about Reed–Muller codes rather than Reed–Solomon codes. The algorithm that we give to solve this task relies on connections to the problem of polynomial identity testing in the area of algebraic complexity theory, as we discuss further below.

Goyal, Ishai, Mahmoody, and Sahai [GIMS10] also study zero knowledge for IPCPs, and show how to obtain IPCPs for $\mathbf{NP}$ that (i) are statistical zero knowledge against unbounded queries, and yet (ii) each location of the (necessarily) super-polynomial size PCP is polynomial-time computable given the NP witness. They further prove that these two properties are not attainable by zero knowledge PCPs. Their construction consists of replacing the commitment scheme in the zero knowledge IP for 3-colorability of [GMW91] with an information-theoretic analogue in the IPCP model. Our Theorem 1.2 also achieves zero knowledge against unbounded queries, but targets the complexity class $\#\mathbf{P}$ (rather than $\mathbf{NP}$), for which there is no clear analogue of property (ii) above.

Information-theoretic commitments also underlie the construction of zero knowledge PCPs [KPT97]. One could apply the [KPT97] result for $\mathbf{NEXP}$ to obtain zero knowledge PCPs (thus also IPCPs) for $\#\mathbf{P}$, but this is an indirect and complex route (in particular, it relies on the PCP Theorem) that, moreover, yields an adaptive honest verifier. Our direct construction is simple and natural, and also yields a non-adaptive honest verifier.

We now discuss the common algebraic structure that allowed us to obtain both of the above results. We believe that further progress in understanding these types of algebraic techniques will lead to further progress in understanding the power of unconditional zero knowledge for IOPs and IPCPs, and perhaps also for MIPs and PCPs.

1.1.3 Succinct constraint detection for Reed–Muller and Reed–Solomon codes

The constructions underlying both of our theorems achieve zero knowledge by applying a simple modification to well-known protocols: the PCP of Ben-Sasson and Sudan [BS08] underlies our result for $\mathbf{NEXP}$ and the sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [LFKN92] underlies our result for $\#\mathbf{P}$.

In both of these protocols the verifier has access (either via a polynomial-size representation or via a PCP oracle) to an exponentially-large word that allegedly belongs to a certain linear code, and the prover 'leaks' hard-to-compute information in the process of convincing the verifier that this word belongs to the linear code. We achieve zero knowledge via a modification that we call *masking*: the prover sends to the verifier a PCP containing a random codeword in this code, and then convinces the verifier that the *sum* of these two (the original codeword and this random codeword) is close to the linear code.[2] Intuitively, zero knowledge comes from the fact that the prover now argues about a random shift of the original word.

However, this idea raises a problem: how does the simulator 'sample' an exponentially-large random codeword in order to answer the verifier's queries to the PCP? Solving this problem crucially relies on solving a problem that lies at the intersection of coding theory, linear algebra, and computational complexity, which we call the *constraint detection problem*. We informally introduce it and state our results about it, and defer to Section 2.2 a more detailed discussion of its connection to zero knowledge.

Detecting constraints in codes. Constraint detection is the problem of determining which linear relations hold across all codewords of a linear code $C \subseteq \mathbb{F}^D$, when considering only a given subdomain $I \subseteq D$ of the code rather than all of the domain D. This problem can always be solved in time that is polynomial in $|D|$ (via Gaussian elimination); however, if there is an algorithm that solves this problem in time that is *polynomial in the subdomain's size* $|I|$, rather than the domain's size $|D|$, then we say that the code has *succinct* constraint detection; in particular, the domain could have *exponential* size and the algorithm would still run in polynomial time.

Definition 1.3 (informal)**.** *We say that a linear code $C \subseteq \mathbb{F}^D$ has* **succinct constraint detection** *if there exists an algorithm that, given a subset $I \subseteq D$, runs in time $\mathrm{poly}(\log|\mathbb{F}| + \log|D| + |I|)$ and outputs $z \in \mathbb{F}^I$ such that $\sum_{i \in I} z(i)w(i) = 0$ for all $w \in C$, or "no" if no such z exists. (In particular, $|D|$ may be exponential.)*

We further discuss the problem of constraint detection in Section 2.1, and provide a formal treatment of it in Section 4.1. Beyond this introduction, we shall use (and achieve) a stronger definition of constraint detection: the algorithm is required to output a basis for the space of dual codewords in $C^\perp$ whose support lies in the subdomain I, i.e., a basis for

[2] This is reminiscent of the use of a random secret share of 0 to achieve privacy in information-theoretic multi-party protocols [BGW88].

the space $\{z \in D^I : \forall\, w \in C\,,\, \sum_{i \in I} z(i)w(i) = 0\}$. Note that in our discussion of succinct constraint detection we do not leverage the distance property of the code C, but we do leverage it in our eventual applications.

Our zero knowledge simulators' strategy includes sampling a "random PCP": a random codeword w in a linear code C with exponentially large domain size $|D|$ (see Section 2.2 for more on this). Explicitly sampling w requires time $\Omega(|D|)$, and so is inefficient. But a verifier makes only polynomially-many queries to w, so the simulator has to only simulate w when restricted to polynomial-size sets $I \subseteq D$, leaving open the possibility of doing so in time $\mathrm{poly}(|I|)$. Achieving such a simulation time is an instance of (efficiently and perfectly) "implementing a huge random object" [GGN10] via a *stateful* algorithm [BW04]. We observe that if C has succinct constraint detection then this sampling problem for C has a solution: the simulator maintains the set $\{(i, a_i)\}_{i \in I}$ of past query-answer pairs; then, on a new verifier query $j \in D$, the simulator uses constraint detection to determine if w_j is linearly dependent on w_I, and answers accordingly (such linear dependencies characterize the required probability distribution, see Lemma 4.3).

Overall, our paper thus provides an application (namely, obtaining zero knowledge simulators) where the problem of efficient implementation of huge random objects arises naturally.

We now state our results about succinct constraint detection.

(1) Reed–Muller codes, and their partial sums. We prove that the family of linear codes comprised of evaluations of low-degree multivariate polynomials, along with their partial sums, has succinct constraint detection. This family is closely related to the *sumcheck protocol* [LFKN92], and indeed we use this result to obtain a PZK analogue of the sumcheck protocol (see Section 2.2.2 and Section 5), which yields Theorem 1.2 (see Section 2.3.1 and Section 6).

Recall that the family of Reed–Muller codes, denoted RM, is indexed by tuples $\mathfrak{n} = (\mathbb{F}, m, d)$, where $\mathbb{F}$ is a finite field and m, d are positive integers, and the $\mathfrak{n}$-th code consists of codewords $w \colon \mathbb{F}^m \to \mathbb{F}$ that are the evaluation of an m-variate polynomial Q of individual degree less than d over $\mathbb{F}$. We denote by $\Sigma\mathrm{RM}$ the family that extends RM with evaluations of all partial sums over certain subcubes of a hypercube:

Definition 1.4 (informal)**.** *We denote by* $\Sigma\mathrm{RM}$ *the linear code family that is indexed by tuples* $\mathfrak{n} = (\mathbb{F}, m, d, H)$*, where* H *is a subset of* $\mathbb{F}$*, and where the* $\mathfrak{n}$*-th code consists of codewords* $(w_0, \ldots, w_m)$ *such that there exists an* m*-variate polynomial* Q *of individual degree less than* d *over* $\mathbb{F}$ *for which* $w_i \colon \mathbb{F}^{m-i} \to \mathbb{F}$ *is the evaluation of the* i*-th partial sum of* Q *over* H*, i.e,* $w_i(\vec{\alpha}) = \sum_{\vec{\gamma} \in H^i} Q(\vec{\alpha}, \vec{\gamma})$ *for every* $\vec{\alpha} \in \mathbb{F}^{m-i}$*.*

The domain size for codes in $\Sigma\mathrm{RM}$ is $\Omega(|\mathbb{F}|^m)$, but our detector's running time is exponentially smaller.

Theorem 1.5 (informal statement of Thm. 4.9)**.** *The family* $\Sigma\mathrm{RM}$ *has succinct constraint detection:*

there is a detector algorithm for $\Sigma\mathrm{RM}$ *that runs in time* $\mathrm{poly}(\log|\mathbb{F}| + m + d + |H| + |I|)$.

We provide intuition for the theorem's proof in Section 2.1.1 and provide the proof's details in Section 4.2; the proof leverages tools from algebraic complexity theory. (Our proof also shows that the family RM, which is a restriction of $\Sigma\mathrm{RM}$, has succinct constraint detection.) Our theorem implies perfect and stateful implementation of a random low-degree multivariate polynomial and its partial sums over any hypercube; our proof extends an algorithm of [BW04], which solves this problem in the case of parity queries to boolean functions on subcubes of the boolean hypercube.

(2) Reed–Solomon codes, and their PCPPs. Second, we prove that the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08] has succinct constraint detection. This family is closely related to quasilinear-size PCPs for $\mathbf{NEXP}$ [BS08], and indeed we use this result to obtain PZK proximity proofs for this family (see Section 2.2.3 and Section 7), from which we derive Theorem 1.1 (see Section 2.3.2 and Section 8).

Definition 1.6 (informal)**.** *We denote by* BS-RS *the linear code family indexed by tuples* $\mathfrak{n} = (\mathbb{F}, L, d)$*, where* $\mathbb{F}$ *is an extension field of* $\mathbb{F}_2$*,* L *is a linear subspace in* $\mathbb{F}$*, and* d *is a positive integer; the* $\mathfrak{n}$*-th code consists of evaluations on* L *of univariate polynomials* Q *of degree less than* d*, concatenated with corresponding [BS08] proximity proofs.*

The domain size for codes in BS-RS is $\Omega(|L|)$, but our detector's running time is exponentially smaller.

Theorem 1.7 (informal statement of Thm. 4.12)**.** *The family* BS-RS *has succinct constraint detection:*

there is a detector algorithm for BS-RS *that runs in time* $\mathrm{poly}(\log|\mathbb{F}| + \dim(L) + |I|)$.

We provide intuition for the theorem's proof in Section 2.1.2 and provide the proof's details in Section 4.3; the proof leverages combinatorial properties of the recursive construction of BS proximity proofs.

2 Techniques

We informally discuss intuition behind our algorithms for detecting constraints (Section 2.1), their connection to zero knowledge (Section 2.2), and how we derive our results about $\#\mathbf{P}$ and $\mathbf{NEXP}$ (Section 2.3). Throughout, we provide pointers to the technical sections that contain further details.

2.1 Detecting constraints for exponentially-large codes

As informally introduced in Section 1.1.3, the *constraint detection problem* corresponding to a linear code family $\mathscr{C} = \{C_{\mathfrak{n}}\}_{\mathfrak{n}}$ with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$ is the following: given an index $\mathfrak{n} \in \{0,1\}^*$ and subset $I \subseteq D(\mathfrak{n})$, output a basis for the space $\{z \in D(\mathfrak{n})^I : \forall\, w \in C_{\mathfrak{n}}\,,\, \sum_{i \in I} z(i)w(i) = 0\}$. In other words, for a given subdomain I, we wish to determine all linear relations that hold for codewords in $C_{\mathfrak{n}}$ restricted to the subdomain I.

If a generating matrix for $C_{\mathfrak{n}}$ can be found in polynomial time, this problem can be solved in $\mathrm{poly}(|\mathfrak{n}|+|D(\mathfrak{n})|)$ time via Gaussian elimination (such an approach was implicitly taken by [BCGV16] to construct a perfect zero knowledge simulator for an IOP for $\mathbf{NP}$). However, in our setting $|D(\mathfrak{n})|$ is *exponential* in $|\mathfrak{n}|$, so the straightforward solution is inefficient. With this in mind, we say that $\mathscr{C}$ has *succinct constraint detection* if there exists an algorithm that solves its constraint detection problem in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time, even if $|D(\mathfrak{n})|$ is exponential in $|\mathfrak{n}|$.

The formal definition of succinct constraint detection is in Section 4.1. In the rest of this section we provide intuition for two of our theorems: succinct constraint detection for the family $\Sigma\mathrm{RM}$ and for the family BS-RS. As will become evident, the techniques that we use to prove the two theorems differ significantly. Perhaps this is because the two codes are quite different: $\Sigma\mathrm{RM}$ has a simple and well-understood algebraic structure, whereas BS-RS is constructed recursively using proof composition.

2.1.1 From algebraic complexity to detecting constraints for Reed–Muller codes and their partial sums

The purpose of this section is to provide intuition about the proof of Theorem 1.5, which states that the family $\Sigma\mathrm{RM}$ has succinct constraint detection. (Formal definitions, statements, and proofs are in Section 4.2.) We thus outline how to construct an algorithm that detects constraints for the family of linear codes comprised of evaluations of low-degree multivariate polynomials, along with their partial sums. Our construction generalizes the proof of [BW04], which solves the special case of parity queries to boolean functions on subcubes of the boolean hypercube by reducing this problem to a probabilistic identity testing problem that is solvable via an algorithm of [RS05].

Below, we temporarily ignore the partial sums, and focus on constructing an algorithm that detects constraints for the family of Reed–Muller codes RM, and at the end of the section we indicate how we can also handle partial sums.

Step 1: phrase as linear algebra problem. Consider a codeword $w\colon \mathbb{F}^m \to \mathbb{F}$ that is the evaluation of an m-variate polynomial Q of individual degree less than d over $\mathbb{F}$. Note that, for every $\vec{\alpha} \in \mathbb{F}^m$, $w(\vec{\alpha})$ equals the inner product of Q's coefficients with the vector $\phi_{\vec{\alpha}}$ that consists of the evaluation of all d^m monomials at $\vec{\alpha}$. One can argue that constraint detection for RM is equivalent to finding the nullspace of $\{\phi_{\vec{\alpha}}\}_{\vec{\alpha} \in I}$. However, "writing out" this $|I| \times d^m$ matrix and performing Gaussian elimination is too expensive, so we must solve this linear algebra problem *succinctly*.

Step 2: encode vectors as coefficients of polynomials. While each vector $\phi_{\vec{\alpha}}$ is long, it has a succinct description; in fact, we can construct an m-variate polynomial $\Phi_{\vec{\alpha}}$ whose coefficients (after expansion) are the entries of $\phi_{\vec{\alpha}}$, but has an arithmetic circuit of only size $O(md)$: namely, $\Phi_{\vec{\alpha}}(\vec{X}) := \prod_{i=1}^{m}(1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1})$. Computing the nullspace of $\{\Phi_{\vec{\alpha}}\}_{\vec{\alpha} \in I}$ is thus equivalent to computing the nullspace of $\{\phi_{\vec{\alpha}}\}_{\vec{\alpha} \in I}$.

Step 3: computing the nullspace. Computing the nullspace of a set of polynomials is a problem in algebraic complexity theory, and is essentially equivalent to the Polynomial Identity Testing (PIT) problem, and so we leverage tools from that area.[3] While there are simple randomized algorithms to solve this problem (see for example [Kay10, Lemma 8] and [BW04]), these algorithms, due to a nonzero probability of error, suffice to achieve statistical zero knowledge but do not suffice to achieve perfect zero knowledge. To obtain perfect zero knowledge, we need a solution that has *no probability of error*. Derandomizing PIT for arbitrary algebraic circuits seems to be beyond current

[3] PIT is the following problem: given a polynomial f expressed as an algebraic circuit, is f identically zero? This problem has well-known randomized algorithms [Zip79, Sch80], but deterministic algorithms for all circuits seem to be beyond current techniques [KI04]. PIT is a central problem in algebraic complexity theory, and suffices for solving a number of other algebraic problems. We refer the reader to [SY10] for a survey.

techniques (as it implies circuit lower bounds [KI04]), but derandomizations are currently known for some restricted circuit classes. The polynomials that we consider are special: they fall in the well-studied class of "sum of products of univariates", and for this case we can invoke the deterministic algorithm of [RS05] (see also [Kay10]). (It is interesting that derandomization techniques are ultimately used to obtain a qualitative improvement for an inherently probabilistic task, i.e., perfect sampling of verifier views.)

The above provides an outline for how to detect constraints for RM. The extension to $\Sigma\mathrm{RM}$, which also includes partial sums, is achieved by considering a more general form of vectors $\phi_{\vec{\alpha}}$ as well as corresponding polynomials $\Phi_{\vec{\alpha}}$. These polynomials also have the special form required for our derandomization. See Section 4.2 for details.

2.1.2 From recursive code covers to detecting constraints for Reed–Solomon codes and their PCPPs

The purpose of this section is to provide intuition about the proof of Theorem 1.7, which states that the family BS-RS has succinct constraint detection. (Formal definitions, statements, and proofs are in Section 4.3.) We thus outline how to construct an algorithm that detects constraints for the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08].

Our construction leverages the recursive structure of BS proximity proofs: we identify key combinatorial properties of the recursion that enable "local" constraint detection. To define and argue these properties, we introduce two notions that play a central role throughout the proof:

> A *(local) view* of a linear code $C \subseteq \mathbb{F}^D$ is a pair $(\tilde{D}, \tilde{C})$ such that $\tilde{D} \subseteq D$ and $\tilde{C} = C|_{\tilde{D}} \subseteq \mathbb{F}^{\tilde{D}}$.
> A *cover* of C is a set of local views $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ of C such that $D = \cup_j \tilde{D}_j$.

Combinatorial properties of the recursive step. Given a finite field $\mathbb{F}$, domain $D \subseteq \mathbb{F}$, and degree d, let $C := \mathrm{RS}[\mathbb{F}, D, d]$ be the Reed–Solomon code consisting of evaluations on D of univariate polynomials of degree less than d over $\mathbb{F}$; for concreteness, say that the domain size is $|D| = 2^n$ and the degree is $d = |D|/2 = 2^{n-1}$.

The first level of [BS08]'s recursion appends to each codeword $f \in C$ an auxiliary function $\pi_1(f)\colon D' \to \mathbb{F}$ with domain D' disjoint from D. Moreover, the mapping from f to $\pi_1(f)$ is linear over $\mathbb{F}$, so the set $C^1 := \{f \| \pi_1(f)\}_{f \in C}$, where $f\|\pi_1(f)\colon D \cup D' \to \mathbb{F}$ is the function that agrees with f on D and with $\pi_1(f)$ on D', is a linear code over $\mathbb{F}$. The code C^1 is the "first-level" code of a BS proximity proof for f.

The code C^1 has a naturally defined cover $S^1 = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ such that each $\tilde{C}_j$ is a Reed–Solomon code $\mathrm{RS}[\mathbb{F}, \tilde{D}_j, d_j]$ with $2d_j \leq |\tilde{D}_j| = O(\sqrt{d})$, that is, with rate $1/2$ and block length $O(\sqrt{d})$. We prove several combinatorial properties of this cover:

- S^1 *is* 1*-intersecting.* For all distinct j, j' in J, $|\tilde{D}_j \cap \tilde{D}_{j'}| \leq 1$ (namely, the subdomains are almost disjoint).
- S^1 *is* $O(\sqrt{d})$*-local.* Every partial assignment to $O(\sqrt{d})$ domains $\tilde{D}_j$ in the cover that is *locally consistent* with the cover can be extended to a *globally consistent* assignment, i.e., to a codeword of C^1. That is, there exists $\kappa = O(\sqrt{d})$ such that every partial assignment $h\colon \cup_{\ell=1}^{\kappa} \tilde{D}_{j_\ell} \to \mathbb{F}$ with $h|_{\tilde{D}_{j_\ell}} \in \tilde{C}_{j_\ell}$ (for each ℓ) equals the restriction to the subdomain $\cup_{\ell=1}^{\kappa} \tilde{D}_{j_\ell}$ of some codeword $f\|\pi_1(f)$ in C^1.
- S^1 *is* $O(\sqrt{d})$*-independent*. The ability to extend locally-consistent assignments to "globally-consistent" codewords of C^1 holds in a stronger sense: even when the aforementioned partial assignment h is extended *arbitrarily* to κ additional point-value pairs, this new partial assignment still equals the restriction of some codeword $f\|\pi_1(f)$ in C^1.

The locality property alone already suffices to imply that, given a subdomain $I \subseteq D \cup D'$ of size $|I| < \sqrt{d}$, we can solve the constraint detection problem on I by considering only those constraints that appear in views that intersect I (see Lemma 4.22). But C has exponential block length so a "quadratic speedup" does not yet imply succinct constraint detection. To obtain it, we also leverage the intersection and independence properties to reduce "locality" as follows.

Further recursive steps. So far we have only considered the first recursive step of a BS proximity proof; we show how to obtain covers with smaller locality (and thereby detect constraints with more efficiency) by considering additional recursive steps. Each code $\tilde{C}_j$ in the cover S^1 of C^1 is a Reed–Solomon code $\mathrm{RS}[\mathbb{F}, \tilde{D}_j, d_j]$ with $|\tilde{D}_j|, d_j = O(\sqrt{d})$, and the next recursive step appends to each codeword in $\tilde{C}_j$ a corresponding auxiliary function, yielding a new code

C^2. In turn, C^2 has a cover S^2, and another recursive step yields a new code C^3, which has its own cover S^3, and so on. The crucial technical observation (Lemma 4.20) is that the intersection and independence properties, which hold recursively, enable us to deduce that C^i is 1-intersecting, $O(\sqrt[2^i]{d})$-local, and $O(\sqrt[2^i]{d})$-independent; in particular, for $r = \log\log d + O(1)$, S^r is 1-intersecting, $O(1)$-local, $O(1)$-independent.

Then, recalling that detecting constraints for local codes requires only the views in the cover that intersect I (Lemma 4.22), our constraint detector works by choosing $i \in \{1, \ldots, r\}$ such that the cover S^i is $\mathrm{poly}(|I|)$-local, finding in this cover a $\mathrm{poly}(|I|)$-size set of $\mathrm{poly}(|I|)$-size views that intersect I, and computing in $\mathrm{poly}(|I|)$ time a basis for the dual of each of these views — thereby proving Theorem 1.7.

Remark 2.1. For the sake of those familiar with BS-RS we remark that the domain D' is the carefully chosen subset of $\mathbb{F} \times \mathbb{F}$ designated by that construction, the code C^1 is the code that evaluates bivariate polynomials of degree $O(\sqrt{d})$ on $D \cup D'$ (along the way mapping $D \subseteq \mathbb{F}$ to a subset of $\mathbb{F} \times \mathbb{F}$), the subdomains $\tilde{D}_j$ are the axis-parallel "rows" and "columns" used in that recursive construction, and the codes $\tilde{C}_j$ are Reed–Solomon codes of block length $O(\sqrt{d})$. The $O(\sqrt{d})$-locality and independence follow from basic properties of bivariate Reed–Muller codes; see Example 4.14.

Remark 2.2. It is interesting to compare the above result with *linear lower bounds on query complexity* for testing proximity to random low density parity check (LDPC) codes [BHR05, BGK+10]. Those results are proved by obtaining a basis for the dual code such that every small-support constraint is spanned by a small subset of that basis. The same can be observed to hold for BS-RS, even though this latter code is locally testable with *polylogarithmic query complexity* [BS08, Thm. 2.13]. The difference between the two cases is due to the fact that, for a random LDPC code, an assignment that satisfies all but a single basis-constraint is (with high probability) far from the code, whereas the recursive and 1-intersecting structure of BS-RS implies the existence of words that satisfy all but a single basis constraint, yet are negligibly close to being a codeword.

2.2 From constraint detection to zero knowledge via masking

We provide intuition about the connection between constraint detection and zero knowledge (Section 2.2.1), and how we leverage this connection to achieve two intermediate results: (i) a sumcheck protocol that is zero knowledge in the Interactive PCP model (Section 2.2.2); and (ii) proximity proofs for Reed–Solomon codes that are zero knowledge in the Interactive Oracle Proof model (Section 2.2.3).

2.2.1 Local simulation of random codewords

Suppose that the prover and verifier both have oracle access to a codeword $w \in C$, for some linear code $C \subseteq \mathbb{F}^D$ with exponential-size domain D, and that they need to engage in some protocol that involves w. During the protocol, the prover may leak information about w that is hard to compute (e.g., requires exponentially-many queries to w), and so would violate zero knowledge (as we see below, this is the case for protocols such as sumcheck).

Rather than directly invoking the protocol, the prover first sends to the verifier a random codeword $r \in C$ (as an oracle since r has exponential size) and the verifier replies with a random field element $\rho \in \mathbb{F}$; then the prover and verifier invoke the protocol on the new codeword $w' := \rho w + r \in C$ rather than w. Intuitively, running the protocol on w' now does not leak information about w, because w' is random in C (up to resolvable technicalities). This *random self-reducibility* makes sense for only some protocols, e.g., those where completeness is preserved for any choice of ρ and soundness is broken for only a small fraction of ρ; but this will indeed be the case for the settings described below.

The aforementioned *masking* technique was used by [BCGV16] for codes with polynomial-size domains, but we use it for codes with exponential-size domains, which requires exponentially more efficient simulation techniques. Indeed, to prove (perfect) zero knowledge, a simulator must be able to reproduce, exactly, the view obtained by any malicious verifier that queries entries of w', a uniformly random codeword in C; however, it is too expensive for the simulator to explicitly sample a random codeword and answer the verifier's queries according to it. Instead, the simulator must sample the "local view" that the verifier sees while querying w' at a *small* number of locations $I \subseteq D$.

But simulating local views of the form $w'|_I$ is reducible to detecting *constraints*, i.e., codewords in the dual code $C^\perp$ whose support is contained in I. Indeed, if no word in $C^\perp$ has support contained in I then $w'|_I$ is uniformly random; otherwise, each additional linearly independent constraint of $C^\perp$ with support contained in I further reduces

the entropy of $w'|_I$ in a well-understood manner. (See Lemma 4.3 for a formal statement.) In sum, succinct constraint detection enables us to "implement" [GGN10, BW04] random codewords of C despite C having exponential size.

Note that in the above discussion we implicitly assumed that the set I is known in advance, i.e., that the verifier chooses its queries in advance. This, of course, need not be the case: a verifier may adaptively make queries based on answers to previous queries and, hence, the set I need not be known a priori. This turns out to not be a problem because, given a constraint detector, it is straightforward to compute the conditional distribution of the view $w'|_I$ given $w'|_J$ for a subset J of I. This is expressed precisely in Lemma 4.3.

We now discuss two concrete protocols for which the aforementioned random self-reducibility applies, and for which we also have constructed suitably-efficient constraint detectors.

2.2.2 Zero knowledge sumchecks

The celebrated sumcheck protocol [LFKN92] is *not* zero knowledge. In the sumcheck protocol, the prover and verifier have oracle access to a low-degree m-variate polynomial F over a field $\mathbb{F}$, and the prover wants to convince the verifier that $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$ for a given subset H of $\mathbb{F}$. During the protocol, the prover communicates partial sums of F, which are $\#\mathbf{P}$-hard to compute and, as such, violate zero knowledge.

We now explain how to use random self-reducibility to make the sumcheck protocol *(perfect) zero knowledge*, at the cost of moving from the Interactive Proof model to the Interactive PCP model.

IPCP sumcheck. Consider the following tweak to the classical sumcheck protocol: rather than invoking sumcheck on F directly, the prover first sends to the verifier (the evaluation of) a random low-degree polynomial R as an oracle; then, the prover sends the value $z := \sum_{\vec{\alpha} \in H^m} R(\vec{\alpha})$ and the verifier replies with a random field element ρ; finally, the two invoke sumcheck on the claim "$\sum_{\vec{\alpha} \in H^m} Q(\vec{\alpha}) = z$" where $Q := \rho F + R$.

Completeness is clear because if $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$ and $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = z$ then $\sum_{\vec{\alpha} \in H^m} (\rho F + R)(\vec{\alpha}) = z$; soundness is also clear because if $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) \neq 0$ then $\sum_{\vec{\alpha} \in H^m} (\rho F + R)(\vec{\alpha}) \neq z$ with high probability over ρ, regardless of the choice of R. (For simplicity, we ignore the fact that the verifier also needs to test that R has low degree.) We are thus left to show (perfect) zero knowledge, which turns out to be a much less straightforward argument.

The simulator. Before we explain how to argue zero knowledge, we first clarify what we mean by it: since the verifier has oracle access to F we cannot hope to 'hide' it; nevertheless, we can hope to argue that the verifier, by participating in the protocol, does not learn anything about F beyond what the verifier can directly learn by querying F (and the fact that F sums to zero on H^m). What we shall achieve is the following: an algorithm that simulates the verifier's view by making as many queries to F as the *total* number of verifier queries to either F or R.[4]

On the surface, zero knowledge seems easy to argue, because $\rho F + R$ seems random among low-degree m-variate polynomials. More precisely, consider the simulator that samples a random low-degree polynomial Q and uses it instead of $\rho F + R$ and answers the verifier queries as follows: (a) whenever the verifier queries $F(\vec{\alpha})$, respond by querying $F(\vec{\alpha})$ and returning the true value; (b) whenever the verifier queries $R(\vec{\alpha})$, respond by querying $F(\vec{\alpha})$ and returning $Q(\vec{\alpha}) - \rho F(\vec{\alpha})$. Observe that the number of queries to F made by the simulator equals the number of (mutually) distinct queries to F and R made by the verifier, as desired.

However, the above reasoning, while compelling, is insufficient. First, $\rho F + R$ is *not* random because a malicious verifier can choose ρ depending on queries to R. Second, even if $\rho F + R$ were random (e.g., the verifier does not query R before choosing ρ), the simulator must run in polynomial time, both producing correctly-distributed 'partial sums' of $\rho F + R$ and answering queries to R, but sampling Q alone requires exponential time. In this high level discussion we ignore the first problem (which nonetheless has to be tackled), and focus on the second.

At this point it should be clear from the discussion in Section 2.2.1 that the simulator does not have to sample Q explicitly, but only has to perfectly simulate local views of it by leveraging the fact that it can keep state across queries. And doing so requires solving the succinct constraint detection problem for a suitable code C. In this case, it suffices to consider the code $C = \Sigma\mathrm{RM}$, and our Theorem 1.5 guarantees the required constraint detector.

The above discussion omits several details, so we refer the reader to Section 5 for further details.

[4] A subsequent work [CFS17] shows how to bootstrap this IPCP sumcheck protocol into a more complex one that has a stronger zero knowledge guarantee: the simulator can sample the verifier's view by making as many queries to F as the number of verifier queries (plus one). Nevertheless, the weaker zero knowledge guarantee that we achieve suffices for our purposes.

2.2.3 Zero knowledge proximity proofs for Reed–Solomon

Testing proximity of a codeword w to a given linear code C can be aided by a *proximity proof* [DR04, BGH$^+$06], which is an auxiliary oracle π that facilitates testing (e.g., C is not locally testable). For example, testing proximity to the Reed–Solomon code, a crucial step towards achieving short PCPs, is aided via suitable proximity proofs [BS08].

From the perspective of zero knowledge, however, a proximity proof can be 'dangerous': a few locations of π can in principle leak a lot of information about the codeword w, and a malicious verifier could potentially learn a lot about w with only a few queries to w and π. The notion of zero knowledge for proximity proofs requires that this cannot happen: it requires the existence of an algorithm that simulates the verifier's view by making as many queries to w as the *total* number of verifier queries to either w or π [IW14]; intuitively, this means that any bit of the proximity proof π reveals no more information than one bit of w.

We demonstrate again the use of random self-reducibility and show a general transformation that, under certain conditions, maps a PCP of proximity (P, V) for a code C to a corresponding 2-round Interactive Oracle Proof of Proximity (IOPP) for C that is *(perfect) zero knowledge*.

IOP of proximity for C. Consider the following IOP of Proximity: the prover and verifier have oracle access to a codeword w, and the prover wants to convince the verifier that w is close to C; the prover first sends to the verifier a random codeword r in C, and the verifier replies with a random field element ρ; the prover then sends the proximity proof $\pi' := P(w')$ that attests that $w' := \rho w + r$ is close to C. Note that this is a 2-round IOP of Proximity for C, because completeness follows from the fact that C is linear, while soundness follows because if w is far from C, then so is $\rho w + r$ for every r with high probability over ρ. But is the zero knowledge property satisfied?

The simulator. Without going into details, analogously to Section 2.2.2, a simulator must be able to sample local views for random codewords from the code $L := \{ w \| P(w) \}_{w \in C}$, so the simulator's efficiency reduces to the efficiency of constraint detection for L. We indeed prove that if L has succinct constraint detection then the simulator works out. See Section 7.1 for further details.

The case of Reed–Solomon. The above machinery allows us to derive a zero knowledge IOP of Proximity for Reed–Solomon codes, thanks to our Theorem 1.7, which states that the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08] has succinct constraint detection; see Section 7.2 for details. This is one of the building blocks of our construction of zero knowledge IOPs for $\mathbf{NEXP}$, as described below in Section 2.3.2.

2.3 Achieving zero knowledge beyond NP

We outline how to derive our results about zero knowledge for $\#\mathbf{P}$ and $\mathbf{NEXP}$.

2.3.1 Zero knowledge for counting problems

We provide intuition for the proof of Theorem 1.2, which states that the complexity class $\#\mathbf{P}$ has Interactive PCPs that are perfect zero knowledge.

We first recall the classical (non zero knowledge) Interactive Proof for $\#\mathbf{P}$ [LFKN92]. The language $\mathscr{L}_{\#3\mathrm{SAT}}$, which consists of pairs (ϕ, N) where ϕ is a 3-CNF boolean formula and N is the number of satisfying assignments of ϕ, is $\#\mathbf{P}$-complete, and thus it suffices to construct an IP for it. The IP for $\mathscr{L}_{\#3\mathrm{SAT}}$ works as follows: the prover and verifier both *arithmetize* ϕ to obtain a low-degree multivariate polynomial p_ϕ and invoke the (non zero knowledge) sumcheck protocol on the claim "$\sum_{\vec{\alpha} \in \{0,1\}^n} p_\phi(\vec{\alpha}) = N$", where arithmetic is over a large-enough prime field.

Returning to our goal, we obtain a perfect zero knowledge Interactive PCP by simply replacing the (non zero knowledge) IP sumcheck mentioned above with our perfect zero knowledge IPCP sumcheck, described in Section 2.2.2. In Section 6 we provide further details, including proving that the zero knowledge guarantees of our sumcheck protocol suffice for this case.

2.3.2 Zero knowledge for nondeterministic time

We provide intuition for the proof of Theorem 1.1, which implies that the complexity class $\mathbf{NEXP}$ has Interactive Oracle Proofs that are perfect zero knowledge. Very informally, the proof consists of combining two building blocks:

(i) [BCGV16]'s reduction from $\mathbf{NEXP}$ to *randomizable* linear algebraic constraint satisfaction problems, and (ii) our construction of perfect zero knowledge IOPs of Proximity for Reed–Solomon codes, described in Section 2.2.3. Besides extending [BCGV16]'s result from $\mathbf{NP}$ to $\mathbf{NEXP}$, our proof provides a conceptual simplification over [BCGV16] by clarifying how the above two building blocks work together towards the final result. We now discuss this.

Starting point: [BS08]. Many PCP constructions consist of two steps: (1) arithmetize the statement at hand (in our case, membership of an instance in some $\mathbf{NEXP}$-complete language) by reducing it to a "PCP-friendly" problem that looks like a *linear-algebraic* constraint satisfaction problem (LACSP); (2) design a tester that probabilistically checks witnesses for this LACSP. In this paper, as in [BCGV16], we take [BS08]'s PCPs for $\mathbf{NEXP}$ as a starting point, where the first step reduces $\mathbf{NEXP}$ to a "univariate" LACSP whose witnesses are codewords in a Reed–Solomon code of exponential degree that satisfy certain properties, and whose second step relies on suitable *proximity proofs* [DR04, BGH$^+$06] for that code. Thus, overall, the PCP consists of two oracles, one being the LACSP witness and the other being the corresponding BS proximity proof, and it is not hard to see that such a PCP is *not* zero knowledge, because both the LACSP witness and its proximity proof reveal hard-to-compute information.

Step 1: sanitize the proximity proof. We first address the problem that the BS proximity proof "leaks", by simply replacing it with our own perfect zero knowledge analogue. Namely, we replace it with our perfect zero knowledge 2-round IOP of Proximity for Reed–Solomon codes, described in Section 2.2.3. This modification ensures that there exists an algorithm that perfectly simulates the verifier's view by making as many queries to the LACSP witness as the *total* number of verifier queries to *either the LACSP witness or other oracles used to facilitate proximity testing*. At this point we have obtained a perfect zero knowledge 2-round IOP of Proximity for $\mathbf{NEXP}$ (analogous to the notion of a zero knowledge PCP of Proximity [IW14]); this part is where, previously, [BCGV16] were restricted to $\mathbf{NP}$ because their simulator only handled Reed–Solomon codes with *polynomial* degree while our simulator is efficient even for such codes with *exponential* degree. But we are not done yet: to obtain our goal, we also need to address the problem that the LACSP witness itself "leaks" when the verifier queries it, which we discuss next.

Step 2: sanitize the witness. Intuitively, we need to inject randomness in the reduction from $\mathbf{NEXP}$ to LACSP because the prover ultimately sends an LACSP witness to the verifier as an oracle, which the verifier can query. This is precisely what [BCGV16]'s reduction from $\mathbf{NEXP}$ to *randomizable* LACSPs enables, and we thus use their reduction to complete our proof. Informally, given an a-priori query bound b on the verifier's queries, the reduction outputs a witness w with the property that one can efficiently sample *another* witness w' whose entries are b-wise independent. We can then simply use the IOP of Proximity from the previous step on this randomized witness. Moreover, since the efficiency of the verifier is polylogarithmic in b, we can set b to be super-polynomial (e.g., exponential) to preserve zero knowledge against any polynomial number of verifier queries.

The above discussion is only a sketch and we refer the reader to Section 8 for further details. One aspect that we did not discuss is that an LACSP witness actually consists of two sub-witnesses, where one is a "local" deterministic function of the other, which makes arguing zero knowledge somewhat more delicate.

2.4 Roadmap

After providing formal definitions in Section 3.1, the rest of the paper is organized as summarized by the table below.

§4.2	**Theorem 1.5/4.9**	detecting constraints for ΣRM	§4.3	**Theorem 1.7/4.12**	detecting constraints for BS-RS
	↓			↓	
§5	**Theorem 5.3**	PZK IPCP for sumcheck	§7	**Theorem 7.3**	PZK IOP of Proximity for RS codes
	↓			↓	
§6	**Theorem 1.2/6.2**	PZK IPCP for $\#\mathbf{P}$	§8	**Theorem 1.1/8.1**	PZK IOP for $\mathbf{NEXP}$

3 Definitions

3.1 Basic notations

Functions, distributions, fields. We use $f\colon D \to R$ to denote a function with domain D and range R; given a subset $\tilde{D}$ of D, we use $f|_{\tilde{D}}$ to denote the restriction of f to $\tilde{D}$. Given a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled according to $\mathcal{D}$. We denote by $\mathbb{F}$ a finite field and by $\mathbb{F}_q$ the field of size q; we say $\mathbb{F}$ is a *binary field* if its characteristic is 2. Arithmetic operations over $\mathbb{F}_q$ cost $\operatorname{polylog} q$ but we shall consider these to have unit cost (and inspection shows that accounting for their actual polylogarithmic cost does not change any of the stated results).

Distances. A distance measure is a function $\Delta\colon \Sigma^n \times \Sigma^n \to [0,1]$ such that for all $x, y, z \in \Sigma^n$: (i) $\Delta(x,x) = 0$, (ii) $\Delta(x,y) = \Delta(y,x)$, and (iii) $\Delta(x,y) \leq \Delta(x,z) + \Delta(z,y)$. We extend Δ to distances to sets: given $x \in \Sigma^n$ and $S \subseteq \Sigma^n$, we define $\Delta(x,S) := \min_{y \in S} \Delta(x,y)$ (or 1 if S is empty). We say that a string x is ϵ-close to another string y if $\Delta(x,y) \leq \epsilon$, and ϵ-far from y if $\Delta(x,y) > \epsilon$; similar terminology applies for a string x and a set S. Unless noted otherwise, we use the *relative Hamming distance* over alphabet Σ (typically implicit): $\Delta(x,y) := |\{i : x_i \neq y_i\}|/n$.

Languages and relations. We denote by $\mathscr{R}$ a (binary ordered) relation consisting of pairs $(\mathtt{x}, \mathtt{w})$, where $\mathtt{x}$ is the *instance* and $\mathtt{w}$ is the *witness*. We denote by $\operatorname{Lan}(\mathscr{R})$ the language corresponding to $\mathscr{R}$, and by $\mathscr{R}|_{\mathtt{x}}$ the set of witnesses in $\mathscr{R}$ for $\mathtt{x}$ (if $\mathtt{x} \notin \operatorname{Lan}(\mathscr{R})$ then $\mathscr{R}|_{\mathtt{x}} := \emptyset$). As always, we assume that $|\mathtt{w}|$ is bounded by some computable function of $n := |\mathtt{x}|$; in fact, we are mainly interested in relations arising from nondeterministic languages: $\mathscr{R} \in \mathbf{NTIME}(T)$ if there exists a $T(n)$-time machine M such that $M(\mathtt{x}, \mathtt{w})$ outputs 1 if and only if $(\mathtt{x}, \mathtt{w}) \in \mathscr{R}$. Throughout, we assume that $T(n) \geq n$. We say that $\mathscr{R}$ has relative distance $\delta_{\mathscr{R}}\colon \mathbb{N} \to [0,1]$ if $\delta_{\mathscr{R}}(n)$ is the minimum relative distance among witnesses in $\mathscr{R}|_{\mathtt{x}}$ for all $\mathtt{x}$ of size n. Throughout, we assume that $\delta_{\mathscr{R}}$ is a constant.

Polynomials. We denote by $\mathbb{F}[X_1, \ldots, X_m]$ the ring of polynomials in m variables over $\mathbb{F}$. Given a polynomial P in $\mathbb{F}[X_1, \ldots, X_m]$, $\deg_{X_i}(P)$ is the degree of P in the variable X_i. We denote by $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ the subspace consisting of $P \in \mathbb{F}[X_1, \ldots, X_m]$ with $\deg_{X_i}(P) < d$ for every $i \in \{1, \ldots, m\}$.

Random shifts. We later use a folklore claim about distance preservation for random shifts in linear spaces.

Claim 3.1. *Let n be in $\mathbb{N}$, $\mathbb{F}$ a finite field, S an $\mathbb{F}$-linear space in $\mathbb{F}^n$, and $x, y \in \mathbb{F}^n$. If x is ϵ-far from S, then $\alpha x + y$ is $\epsilon/2$-far from S, with probability $1 - |\mathbb{F}|^{-1}$ over a random $\alpha \in \mathbb{F}$. (Distances are relative Hamming distances.)*

3.2 Single-prover proof systems

We use two types of proof systems that combine aspects of interactive proofs [Bab85, GMR89] and probabilistically checkable proofs [BFLS91, AS98, ALM+98]: **interactive PCPs** (IPCPs) [KR08] and **interactive oracle proofs** (IOPs) [BCS16, RRR16]. We first describe IPCPs (Section 3.2.1) and then IOPs (Section 3.2.2), which generalize the former.

3.2.1 Interactive probabilistically checkable proofs

An **IPCP** [KR08] is a PCP followed by an IP. Namely, the prover P and verifier V interact as follows: P sends to V a probabilistically checkable proof π; afterwards, P and V^π engage in an interactive proof. Thus, V may read a few bits of π but must read subsequent messages from P in full. An *IPCP system* for a relation $\mathscr{R}$ is thus a pair (P, V), where P, V are probabilistic interactive algorithms working as described, that satisfies naturally-defined notions of perfect completeness and soundness with a given error $\varepsilon(\cdot)$; see [KR08] for details.

We say that an IPCP has k rounds if this "PCP round" is followed by a $(\mathsf{k}-1)$-round interactive proof. (That is, we count the PCP round towards round complexity, unlike [KR08].) Beyond round complexity, we also measure how many bits the prover sends and how many the verifier reads: the *proof length* l is the length of π in bits plus the number of bits in all subsequent prover messages; the *query complexity* q is the number of bits of π read by the verifier plus the number of bits in all subsequent prover messages (since the verifier must read all of those bits).

In this work, we do not count the number of bits in the verifier messages, nor the number of random bits used by the verifier; both are bounded from above by the verifier's running time, which we do consider. Overall, we say that a relation $\mathscr{R}$ belongs to the complexity class $\mathbf{IPCP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \mathsf{tp}, \mathsf{tv}]$ if there is an IPCP system for $\mathscr{R}$ in which: (1) the number of rounds is at most $\mathsf{k}(n)$; (2) the proof length is at most $\mathsf{l}(n)$; (3) the query complexity is at most $\mathsf{q}(n)$; (4) the soundness error is $\varepsilon(n)$; (5) the prover algorithm runs in time $\mathsf{tp}(n)$; (6) the verifier algorithm runs in time $\mathsf{tv}(n)$.

3.2.2 Interactive oracle proofs

An **IOP** [BCS16, RRR16] is a "multi-round PCP". That is, an IOP generalizes an interactive proof as follows: whenever the prover sends to the verifier a message, the verifier does not have to read the message in full but may probabilistically query it. In more detail, a k-round IOP comprises k rounds of interaction. In the i-th round of interaction: the verifier sends a message m_i to the prover; then the prover replies with a message π_i to the verifier, which the verifier can query in this and later rounds (via oracle queries). After the k rounds of interaction, the verifier either accepts or rejects.

An *IOP system* for a relation $\mathscr{R}$ with soundness error ε is thus a pair (P, V), where P, V are probabilistic interactive algorithms working as described, that satisfies the following properties. (See [BCS16] for more details.)

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathscr{R}$, $\Pr[\langle P(\mathtt{x}, \mathtt{w}), V(\mathtt{x})\rangle = 1] = 1$.

Soundness: For every instance $\mathtt{x}$ not in $\mathscr{R}$'s language and unbounded malicious prover $\tilde{P}$, $\Pr[\langle \tilde{P}, V(\mathtt{x})\rangle = 1] \leq \varepsilon(n)$.

Beyond round complexity, we also measure how many bits the prover sends and how many the verifier reads: the *proof length* l is the total number of bits in all of the prover's messages, and the *query complexity* q is the total number of bits read by the verifier across all of the prover's messages. Considering all of these parameters, we say that a relation $\mathscr{R}$ belongs to the complexity class $\mathbf{IOP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \mathsf{tp}, \mathsf{tv}]$ if there is an IOP system for $\mathscr{R}$ in which: (1) the number of rounds is at most $\mathsf{k}(n)$; (2) the proof length is at most $\mathsf{l}(n)$; (3) the query complexity is at most $\mathsf{q}(n)$; (4) the soundness error is $\varepsilon(n)$; (5) the prover algorithm runs in time $\mathsf{tp}(n)$; (6) the verifier algorithm runs in time $\mathsf{tv}(n)$.

IOP vs. IPCP. An IPCP (see Section 3.2.1) is a special case of an IOP because an IPCP verifier must read in full all of the prover's messages except the first one (while an IOP verifier may query any part of any prover message). The above complexity measures are consistent with those defined for IPCPs.

3.2.3 Restrictions and extensions

The definitions below are about IOPs, but IPCPs inherit all of these definitions because they are a special case of IOP.

Adaptivity of queries. An IOP system is *non-adaptive* if the verifier queries are non-adaptive, i.e., the queried locations depend only on the verifier's inputs.

Public coins. An IOP system is *public coin* if each verifier message m_i is chosen uniformly and independently at random, and all of the verifier queries happen after receiving the last prover message.

Proximity. An *IOP of proximity* extends the definition of an IOP in the same way that a PCP of proximity extends that of a PCP [DR04, BGH$^+$06]. An *IOPP system* for a relation $\mathscr{R}$ with 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 $\mathscr{R}$, $\Pr[\langle P(\mathtt{x}, \mathtt{w}), V^{\mathtt{w}}(\mathtt{x})\rangle = 1] = 1$.

Soundness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ with $\Delta(\mathtt{w}, \mathscr{R}|_{\mathtt{x}}) \geq \delta(n)$ and unbounded malicious prover $\tilde{P}$, $\Pr[\langle \tilde{P}, V^{\mathtt{w}}(\mathtt{x})\rangle = 1] \leq \varepsilon(n)$.

Similarly to above, a relation $\mathscr{R}$ belongs to the complexity class $\mathbf{IOPP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \delta, \mathsf{tp}, \mathsf{tv}]$ if there is an IOPP system for $\mathscr{R}$ with the corresponding parameters. Following [IW14], we call an IOPP *exact* if $\delta(n) = 0$.

Promise relations. A *promise relation* is a relation-language pair $(\mathscr{R}^{\mathsf{YES}}, \mathscr{L}^{\mathsf{NO}})$ with $\mathrm{Lan}(\mathscr{R}^{\mathsf{YES}}) \cap \mathscr{L}^{\mathsf{NO}} = \emptyset$. An IOP for a promise relation is the same as an IOP for the (standard) relation $\mathscr{R}^{\mathsf{YES}}$, except that soundness need only hold for $\mathtt{x} \in \mathscr{L}^{\mathsf{NO}}$. An IOPP for a promise relation is the same as an IOPP for the (standard) relation $\mathscr{R}^{\mathsf{YES}}$, except that soundness need only hold for $\mathtt{x} \in \mathrm{Lan}(\mathscr{R}^{\mathsf{YES}}) \cup \mathscr{L}^{\mathsf{NO}}$.

3.2.4 Prior constructions

In this paper we give new IPCP and IOP constructions that achieve perfect zero knowledge for various settings. Below we summarize known constructions in these two models.

IPCPs. Prior work obtains IPCPs with proof length that depends on the witness size rather than computation size [KR08, GKR08], and IPCPs with statistical zero knowledge [GIMS10] (see Section 3.3 for more details).

IOPs. Prior work obtains IOPs with perfect zero knowledge for $\mathbf{NP}$ [BCGV16], IOPs with small proof length and query complexity [BCG+17], and an amortization theorem for "unambiguous" IOPs [RRR16]. Also, [BCS16] show how to compile public-coin IOPs into non-interactive arguments in the random oracle model.

3.3 Zero knowledge

We define the notion of zero knowledge for IOPs and IPCPs achieved by our constructions: *unconditional (perfect) zero knowledge via straightline simulators*. This notion is quite strong not only because it unconditionally guarantees simulation of the verifier's view but also because straightline simulation implies desirable properties such as composability. We now provide some context and then give formal definitions.

At a high level, zero knowledge requires that the verifier's view can be efficiently simulated without the prover. Converting the informal statement into a mathematical one involves many choices, including choosing which verifier class to consider (e.g., the honest verifier? all polynomial-time verifiers?), the quality of the simulation (e.g., is it identically distributed to the view? statistically close to it? computationally close to it?), the simulator's dependence on the verifier (e.g., is it non-uniform? or is the simulator universal?), and others. The definitions below consider two variants: perfect simulation via universal simulators against either unbounded-query or bounded-query verifiers.

Moreover, in the case of universal simulators, one distinguishes between a non-blackbox use of the verifier, which means that the simulator takes the verifier's code as input, and a blackbox use of it, which means that the simulator only accesses the verifier via a restricted interface; we consider this latter case. Different models of proof systems call for different interfaces, which grant carefully-chosen "extra powers" to the simulator (in comparison to the prover) so to ensure that efficiency of the simulation does not imply the ability to efficiently decide the language. For example: in ZK IPs, the simulator may rewind the verifier; in ZK PCPs, the simulator may adaptively answer oracle queries. In ZK IPCPs and ZK IOPs (our setting), the natural definition would allow a blackbox simulator to rewind the verifier *and also* to adaptively answer oracle queries. The definitions below, however, consider only simulators that are straightline [FS89, DS98], that is they do not rewind the verifier, because our constructions achieve this stronger notion.

We are now ready to define the notion of unconditional (perfect) zero knowledge via straightline simulators. We first discuss the notion for IOPs, then for IOPs of proximity, and finally for IPCPs.

3.3.1 ZK for IOPs

We define zero knowledge (via straightline simulators) for IOPs. We begin by defining the view of an IOP verifier.

Definition 3.2. *Let A, B be algorithms and x, y strings. We denote by* $\mathrm{View}\ \langle B(y), A(x)\rangle$ *the* **view** *of $A(x)$ in an interactive oracle protocol with $B(y)$, i.e., the random variable $(x, r, a_1, \ldots, a_n)$ where x is A's input, r is A's randomness, and $a_1, \ldots, a_n$ are the answers to A's queries into B's messages.*

Straightline simulators in the context of IPs were used in [FS89], and later defined in [DS98]. The definition below considers this notion in the context of IOPs, where the simulator also has to answer oracle queries by the verifier. Note that since we consider the notion of unconditional (perfect) zero knowledge, the definition of straightline simulation needs to allow the efficient simulator to work even with inefficient verifiers [GIMS10].

Definition 3.3. *We say that an algorithm B has* **straightline access** *to another algorithm A if B interacts with A, without rewinding, by exchanging messages with A and also answering any oracle queries along the way. We denote by B^A the concatenation of A's random tape and B's output. (Since A's random tape could be super-polynomially large, B cannot sample it for A and then output it; instead, we restrict B to not see it, and we prepend it to B's output.)*

Recall that an algorithm A is b-*query* if, on input $\mathtt{x}$, it makes at most $\mathsf{b}(|\mathtt{x}|)$ queries to any oracles it has access to. We are now ready to define zero knowledge IOPs.

Definition 3.4. *An IOP system (P, V) for a relation $\mathscr{R}$ is perfect zero knowledge (via straightline simulators) against unbounded queries (resp., against query bound b) if there exists a simulator algorithm S such that for every algorithm (resp., b-query algorithm) $\tilde{V}$ and instance-witness pair $(\mathtt{x}, \mathtt{w}) \in \mathscr{R}$, $S^{\tilde{V}}(\mathtt{x})$ and $\mathrm{View}\ \langle P(\mathtt{x}, \mathtt{w}), \tilde{V}(\mathtt{x})\rangle$ are identically distributed. Moreover, S must run in time $\mathrm{poly}(|\mathtt{x}| + \mathsf{q}_{\tilde{V}}(|\mathtt{x}|))$, where $\mathsf{q}_{\tilde{V}}(\cdot)$ is $\tilde{V}$'s query complexity.*

For zero knowledge against arbitrary polynomial-time adversaries, it suffices for b to be superpolynomial. Note that S's running time need not be polynomial in b (in our constructions it is polylogarithmic in b); rather its running time may be polynomial in the input size $|\mathbb{x}|$ and the *actual* number of queries $\tilde{V}$ makes (as a random variable).

We say that a relation $\mathscr{R}$ belongs to the complexity class $\mathbf{PZK\text{-}IOP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \mathsf{tp}, \mathsf{tv}, \mathsf{b}]$ if there is an IOP system for $\mathscr{R}$, with the corresponding parameters, that is perfect zero knowledge with query bound b; also, it belongs to the complexity class $\mathbf{PZK\text{-}IOP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \mathsf{tp}, \mathsf{tv}, *]$ if the same is true with unbounded queries.

3.3.2 ZK for IOPs of proximity

We define zero knowledge (via straightline simulators) for IOPs of proximity. It is a straightforward extension of the corresponding notion for PCPs of proximity, introduced in [IW14].

Definition 3.5. *An IOPP system (P, V) for a relation $\mathscr{R}$ is perfect zero knowledge (via straightline simulators) against unbounded queries (resp., against query bound b) if there exists a simulator algorithm S such that for every algorithm (resp., b-query algorithm) $\tilde{V}$ and instance-witness pair $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$, the following two random variables are identically distributed:*

$$\left(S^{\tilde{V}, \mathbb{w}}(\mathbb{x}) \ , \ q_S\right) \quad \textit{and} \quad \left(\mathrm{View}\ \langle P(\mathbb{x}, \mathbb{w}), \tilde{V}^{\mathbb{w}}(\mathbb{x})\rangle \ , \ q_{\tilde{V}}\right) \ ,$$

where q_S is the number of queries to $\mathbb{w}$ made by S, and $q_{\tilde{V}}$ is the number of queries to $\mathbb{w}$ or to prover messages made by $\tilde{V}$. Moreover, S must run in time $\mathrm{poly}(|\mathbb{x}| + \mathsf{q}_{\tilde{V}}(|\mathbb{x}|))$, where $\mathsf{q}_{\tilde{V}}(\cdot)$ is $\tilde{V}$'s query complexity.

We say that a relation $\mathscr{R}$ belongs to the complexity class $\mathbf{PZK\text{-}IOPP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \delta, \mathsf{tp}, \mathsf{tv}, \mathsf{b}]$ if there is an IOPP system for $\mathscr{R}$, with the corresponding parameters, that is perfect zero knowledge with query bound b; also, it belongs to the complexity class $\mathbf{PZK\text{-}IOPP}[\mathsf{k}, \mathsf{l}, \mathsf{q}, \varepsilon, \delta, \mathsf{tp}, \mathsf{tv}, *]$ if the same is true with unbounded queries.

Remark 3.6. Analogously to [IW14], our definition of zero knowledge for IOPs of proximity requires that the number of queries to $\mathbb{w}$ by S equals the total number of queries (to $\mathbb{w}$ or prover messages) by $\tilde{V}$. Stronger notions are possible: "the number of queries to $\mathbb{w}$ by S equals the number of queries to $\mathbb{w}$ by $\tilde{V}$"; or, even more, "S and $\tilde{V}$ read the same locations of $\mathbb{w}$". The definition above is sufficient for the applications of IOPs of proximity that we consider.

3.3.3 ZK for IPCPs

The definition of perfect zero knowledge (via straightline simulators) for IPCPs follows directly from Definition 3.4 in Section 3.3.1 because IPCPs are a special case of IOPs. Ditto for IPCPs of proximity, whose perfect zero knowledge definition follows directly from Definition 3.5 in Section 3.3.2. (For comparison, [GIMS10] define statistical zero knowledge IPCPs, also with straightline simulators.)

3.4 Codes

An error correcting code C is a set of functions $w\colon D \to \Sigma$, where D, Σ are finite sets known as the domain and alphabet; we write $C \subseteq \Sigma^D$. The message length of C is $k := \log_{|\Sigma|} |C|$, its block length is $\ell := |D|$, its rate is $\rho := k/\ell$, its (minimum) distance is $d := \min\{\Delta(w, z) \ : \ w, z \in C \ , \ w \neq z\}$ when Δ is the (absolute) Hamming distance, and its (minimum) relative distance is $\tau := d/\ell$. At times we write $k(C), \ell(C), \rho(C), d(C), \tau(C)$ to make the code under consideration explicit. All the codes we consider are linear codes, discussed next.

Linearity. A code C is *linear* if Σ is a finite field and C is a Σ-linear space in Σ^D. The dual code of C is the set $C^\perp$ of functions $z\colon D \to \Sigma$ such that, for all $w\colon D \to \Sigma$, $\langle z, w\rangle := \sum_{i \in D} z(i)w(i) = 0$. We denote by $\dim(C)$ the dimension of C; it holds that $\dim(C) + \dim(C^\perp) = \ell$ and $\dim(C) = k$ (dimension equals message length).

Code families. A code family $\mathscr{C} = \{C_{\mathbb{n}}\}_{\mathbb{n} \in \{0,1\}^*}$ has domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$ if each code $C_{\mathbb{n}}$ has domain $D(\mathbb{n})$ and alphabet $\mathbb{F}(\mathbb{n})$. Similarly, $\mathscr{C}$ has message length $k(\cdot)$, block length $\ell(\cdot)$, rate $\rho(\cdot)$, distance $d(\cdot)$, and relative distance $\tau(\cdot)$ if each code $C_{\mathbb{n}}$ has message length $k(\mathbb{n})$, block length $\ell(\mathbb{n})$, rate $\rho(\mathbb{n})$, distance $d(\mathbb{n})$, and relative distance $\tau(\mathbb{n})$. We also define $\rho(\mathscr{C}) := \inf_{\mathbb{n} \in \mathbb{N}} \rho(\mathbb{n})$ and $\tau(\mathscr{C}) := \inf_{\mathbb{n} \in \mathbb{N}} \tau(\mathbb{n})$.

Reed–Solomon codes. The Reed–Solomon (RS) code is the code consisting of evaluations of *univariate* low-degree polynomials: given a field $\mathbb{F}$, subset S of $\mathbb{F}$, and positive integer d with $d \leq |S|$, we denote by $\mathrm{RS}[\mathbb{F}, S, d]$ the linear

code consisting of evaluations $w\colon S \to \mathbb{F}$ over S of polynomials in $\mathbb{F}^{<d}[X]$. The code's message length is $k = d$, block length is $\ell = |S|$, rate is $\rho = \frac{d}{|S|}$, and relative distance is $\tau = 1 - \frac{d-1}{|S|}$.

Reed–Muller codes. The Reed–Muller (RM) code is the code consisting of evaluations of *multivariate* low-degree polynomials: given a field $\mathbb{F}$, subset S of $\mathbb{F}$, and positive integers m, d with $d \leq |S|$, we denote by $\mathrm{RM}[\mathbb{F}, S, m, d]$ the linear code consisting of evaluations $w\colon S^m \to \mathbb{F}$ over S^m of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ (i.e., we bound individual degrees rather than their sum). The code's message length is $k = d^m$, block length is $\ell = |S|^m$, rate is $\rho = (\frac{d}{|S|})^m$, and relative distance is $\tau = (1 - \frac{d-1}{|S|})^m$.

4 Succinct constraint detection

We introduce the notion of *succinct constraint detection* for linear codes. This notion plays a crucial role in constructing perfect zero knowledge simulators for super-polynomial complexity classes (such as $\#\mathbf{P}$ and $\mathbf{NEXP}$), but we believe that this naturally-defined notion is also of independent interest. Given a linear code $C \subseteq \mathbb{F}^D$ we refer to its dual code $C^\perp \subseteq \mathbb{F}^D$ as the *constraint space* of C. The *constraint detection problem* corresponding to a family of linear codes $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$ is the following:

Given an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, output a basis for $\{z \in D(\mathfrak{n})^I : \forall\, w \in C_\mathfrak{n}\,,\, \sum_{i \in I} z(i)w(i) = 0\}$.[5]

If $|D(\mathfrak{n})|$ is polynomial in $|\mathfrak{n}|$ and a generating matrix for $C_\mathfrak{n}$ can be found in polynomial time, this problem can be solved in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time via Gaussian elimination; such an approach was implicitly taken by [BCGV16] to construct a perfect zero knowledge simulator for an IOP for $\mathbf{NP}$. However, in our setting, $|D(\mathfrak{n})|$ is *exponential* in $|\mathfrak{n}|$ and $|I|$, and the aforementioned generic solution requires exponential time. With this in mind, we say $\mathscr{C}$ has *succinct constraint detection* if there exists an algorithm that solves the constraint detection problem in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time when $|D(\mathfrak{n})|$ is *exponential* in $|\mathfrak{n}|$. After defining succinct constraint detection in Section 4.1, we proceed as follows.

- In Section 4.2, we construct a succinct constraint detector for the family of linear codes comprised of evaluations of partial sums of low-degree polynomials. The construction of the detector exploits derandomization techniques from algebraic complexity theory. Later on (in Section 5), we leverage this result to construct a perfect zero knowledge simulator for an IPCP for $\#\mathbf{P}$.
- In Section 4.3, we construct a succinct constraint detector for the family of evaluations of univariate polynomials concatenated with corresponding BS proximity proofs [BS08]. The construction of the detector exploits the recursive structure of these proximity proofs. Later on (in Section 8), we leverage this result to construct a perfect zero knowledge simulator for an IOP for $\mathbf{NEXP}$; this simulator can be interpreted as an analogue of [BCGV16]'s simulator that runs *exponentially faster* and thus enables us to "scale up" from $\mathbf{NP}$ to $\mathbf{NEXP}$.

Throughout this section we assume familiarity with terminology and notation about codes, introduced in Section 3.4. We assume for simplicity that $|\mathfrak{n}|$, the number of bits used to represent $\mathfrak{n}$, is at least $\log D(\mathfrak{n}) + \log \mathbb{F}(\mathfrak{n})$; if this does not hold, then one can replace $|\mathfrak{n}|$ with $|\mathfrak{n}| + \log D(\mathfrak{n}) + \log \mathbb{F}(\mathfrak{n})$ throughout the section.

Remark 4.1 (sparse representation)**.** In this section we make statements about vectors v in $\mathbb{F}^D$ where the cardinality of the domain D may be super-polynomial. When such statements are computational in nature, we assume that v is not represented as a list of $|D|$ field elements (which requires $\Omega(|D| \log |\mathbb{F}|)$ bits) but, instead, assume that v is represented as a list of the elements in $\mathrm{supp}(v)$ (and each element comes with its index in D); this *sparse* representation only requires $\Omega(|\mathrm{supp}(v)| \cdot (\log |D| + \log |\mathbb{F}|))$ bits.

4.1 Definition of succinct constraint detection

Formally define the notion of a *constraint detector*, and the notion of *succinct constraint detection*.

Definition 4.2. *Let $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. A* **constraint detector** *for $\mathscr{C}$ is an algorithm that, on input an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, outputs a basis for the space*

$$\left\{ z \in D(\mathfrak{n})^I : \forall\, w \in C_\mathfrak{n}\,, \sum_{i \in I} z(i)w(i) \right\}\,.$$

*We say that $\mathscr{C}$ has $T(\cdot,\cdot)$-***time constraint detection** *if there exists a detector for $\mathscr{C}$ running in time $T(\mathfrak{n}, \ell)$; we also say that $\mathscr{C}$ has* **succinct constraint detection** *if it has $\mathrm{poly}(|\mathfrak{n}| + \ell)$-time constraint detection.*

A constraint detector induces a corresponding probabilistic algorithm for 'simulating' answers to queries to a random codeword; this is captured by the following lemma, the proof of which is in Appendix B. We shall use such probabilistic algorithms in the construction of perfect zero knowledge simulators (see Section 5 and Section 8).

[5] In fact, the following weaker definition suffices for the applications in our paper: *given an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, output $z \in \mathbb{F}(\mathfrak{n})^I$ such that $\sum_{i \in I} z(i)w(i) = 0$ for all $w \in C_\mathfrak{n}$, or 'independent' if no such z exists.* We achieve the stronger definition, which is also easier to work with.

Lemma 4.3. *Let $\mathscr{C} = \{C_{\mathbb{n}}\}_{\mathbb{n}}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$ that has $T(\cdot,\cdot)$-time constraint detection. Then there exists a probabilistic algorithm $\mathcal{A}$ such that, for every index $\mathbb{n}$, set of pairs $S = \{(\alpha_1,\beta_1),\ldots,(\alpha_\ell,\beta_\ell)\} \subseteq D(\mathbb{n}) \times \mathbb{F}(\mathbb{n})$, and pair $(\alpha,\beta) \in D(\mathbb{n}) \times \mathbb{F}(\mathbb{n})$,*

$$\Pr\Big[\mathcal{A}(\mathbb{n},S,\alpha) = \beta\Big] = \Pr_{w \leftarrow C_{\mathbb{n}}}\left[w(\alpha) = \beta \;\middle|\; \begin{array}{c} w(\alpha_1) = \beta_1 \\ \vdots \\ w(\alpha_\ell) = \beta_\ell \end{array} \right] .$$

Moreover $\mathcal{A}$ runs in time $T(\mathbb{n},\ell) + \mathrm{poly}(\log|\mathbb{F}(\mathbb{n})| + \ell)$.

For the purposes of *constructing* a constraint detector, the sufficient condition given in Lemma 4.6 below is sometimes easier to work with. To state it we need to introduce two ways of restricting a code, and explain how these restrictions interact with taking duals; the interplay between these is delicate (see Remark 4.7).

Definition 4.4. *Given a linear code $C \subseteq \mathbb{F}^D$ and a subset $I \subseteq D$, we denote by (i) $C_{\subseteq I}$ the set consisting of the codewords $w \in C$ for which $\mathrm{supp}(w) \subseteq I$, and (ii) $C|_I$ the restriction to I of codewords $w \in C$.*

Note that $C_{\subseteq I}$ and $C|_I$ are *different notions*. Consider for example the 1-dimensional linear code $C = \{00, 11\}$ in $\mathbb{F}_2^{\{1,2\}}$ and the subset $I = \{1\}$: it holds that $C_{\subseteq I} = \{00\}$ and $C|_I = \{0,1\}$. In particular, codewords in $C_{\subseteq I}$ are defined over D, while codewords in $C|_I$ are defined over I. Nevertheless, throughout this section, we sometimes compare vectors defined over different domains, with the implicit understanding that the comparison is conducted over the union of the relevant domains, by filling in zeros in the vectors' undefined coordinates. For example, we may write $C_{\subseteq I} \subseteq C|_I$ to mean that $\{00\} \subseteq \{00, 10\}$ (the set obtained from $\{0,1\}$ after filling in the relevant zeros).

Claim 4.5. *Let C be a linear code with domain D and alphabet $\mathbb{F}$. For every $I \subseteq D$,*

$$(C|_I)^\perp = (C^\perp)_{\subseteq I} \ ,$$

that is,

$$\Big\{ z \in D(\mathbb{n})^I \,:\, \forall\, w \in C_{\mathbb{n}} \,, \sum_{i \in I} z(i)w(i) \Big\} = \Big\{ z \in C_{\mathbb{n}}^\perp : \mathrm{supp}(z) \subseteq I \Big\} \ .$$

Proof. For the containment $(C^\perp)_{\subseteq I} \subseteq (C|_I)^\perp$: if $z \in C^\perp$ and $\mathrm{supp}(z) \subseteq I$ then z lies in the dual of $C|_I$ because it suffices to consider the subdomain I for determining duality. For the reverse containment $(C^\perp)_{\subseteq I} \supseteq (C|_I)^\perp$: if $z \in (C|_I)^\perp$ then $\mathrm{supp}(z) \subseteq I$ (by definition) so that $\langle z, w\rangle = \langle z, w|_I\rangle$ for every $w \in C$, and the latter inner product equals 0 because z is in the dual of $C|_I$; in sum z is dual to (all codewords in) C and its support is contained in I, so z belongs to $(C^\perp)_{\subseteq I}$, as claimed. □

Observe that Claim 4.5 tells us the constraint detection is equivalent to determining a basis of $(C_{\mathbb{n}}|_I)^\perp = (C_{\mathbb{n}}^\perp)_{\subseteq I}$. The following lemma asserts that if, given a subset $I \subseteq D$, we can find a set of constraints W in $C^\perp$ that spans $(C^\perp)_{\subseteq I}$ then we can solve the constraint detection problem for C; we defer the proof of the lemma to Appendix C.

Lemma 4.6. *Let $\mathscr{C} = \{C_{\mathbb{n}}\}_{\mathbb{n}}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. If there exists an algorithm that, on input an index $\mathbb{n}$ and subset $I \subseteq D(\mathbb{n})$, outputs in $\mathrm{poly}(|\mathbb{n}| + |I|)$ time a subset $W \subseteq \mathbb{F}(\mathbb{n})^{D(\mathbb{n})}$ (in sparse representation) with $(C_{\mathbb{n}}^\perp)_{\subseteq I} \subseteq \mathrm{span}(W) \subseteq C_{\mathbb{n}}^\perp$, then $\mathscr{C}$ has succinct constraint detection.*

Remark 4.7. The following operations do *not* commute: (i) expanding the domain via zero padding (for the purpose of comparing vectors over different domains), and (ii) taking the dual of the code. Consider for example the code $C = \{0\} \subseteq \mathbb{F}_2^{\{1\}}$: its dual code is $C^\perp = \{0,1\}$ and, when expanded to $\mathbb{F}_2^{\{1,2\}}$, the dual code is expanded to $\{(0,0),(1,0)\}$; yet, when C is expanded to $\mathbb{F}_2^{\{1,2\}}$ it produces the code $\{(0,0)\}$ and its dual code is $\{(0,0),(1,0),(0,1),(1,1)\}$. To resolve ambiguities (when asserting an equality as in Claim 4.5), we adopt the convention that expansion is done *always last* (namely, as late as possible without having to compare vectors over different domains).

4.2 Partial sums of low-degree polynomials

We show that evaluations of partial sums of low-degree polynomials have succinct constraint detection (see Definition 4.2). In the following, $\mathbb{F}$ is a finite field, m, d are positive integers, and H is a subset of $\mathbb{F}$; also, $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ denotes the subspace of $\mathbb{F}[X_1, \ldots, X_m]$ consisting of those polynomials with individual degrees less than d. Moreover, given $Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ and $\vec{\alpha} \in \mathbb{F}^{\leq m}$ (vectors over $\mathbb{F}$ of length at most m), we define $Q(\vec{\alpha}) := \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} Q(\vec{\alpha}, \vec{\gamma})$, i.e., the answer to a query that specifies only a suffix of the variables is the sum of the values obtained by letting the remaining variables range over H. We begin by defining the code that we study, which extends the Reed–Muller code (see Section 3.4) with partial sums.

Definition 4.8. *We denote by $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$ the linear code that comprises evaluations of partial sums of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$; more precisely, $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H] := \{w_Q\}_{Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m]}$ where $w_Q \colon \mathbb{F}^{\leq m} \to \mathbb{F}$ is the function defined by $w_Q(\vec{\alpha}) := \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} Q(\vec{\alpha}, \vec{\gamma})$ for each $\vec{\alpha} \in \mathbb{F}^{\leq m}$.[6] We denote by $\Sigma\mathrm{RM}$ the linear code family indexed by tuples $\mathfrak{n} = (\mathbb{F}, m, d, H)$ and where the $\mathfrak{n}$-th code equals $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$. (We represent indices $\mathfrak{n}$ so to ensure that $|\mathfrak{n}| = \Theta(\log |\mathbb{F}| + m + d + |H|)$.)*

We prove that the linear code family $\Sigma\mathrm{RM}$ has succinct constraint detection:

Theorem 4.9 (formal statement of 1.5)**.** *$\Sigma\mathrm{RM}$ has $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H| + \ell)$-time constraint detection.*

Combined with Lemma 4.3, the theorem above implies that there exists a probabilistic polynomial-time algorithm for answering queries to a codeword sampled at random from $\Sigma\mathrm{RM}$, as captured by the following corollary.

Corollary 4.10. *There exists a probabilistic algorithm $\mathcal{A}$ such that, for every finite field $\mathbb{F}$, positive integers m, d, subset H of $\mathbb{F}$, subset $S = \{(\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)\} \subseteq \mathbb{F}^{\leq m} \times \mathbb{F}$, and $(\alpha, \beta) \in \mathbb{F}^{\leq m} \times \mathbb{F}$,*

$$\Pr\Big[\mathcal{A}(\mathbb{F}, m, d, H, S, \alpha) = \beta\Big] = \Pr_{R \leftarrow \mathbb{F}^{<d}[X_1, \ldots, X_m]}\left[R(\alpha) = \beta \;\middle|\; \begin{array}{c} R(\alpha_1) = \beta_1 \\ \vdots \\ R(\alpha_\ell) = \beta_\ell \end{array}\right] .$$

Moreover $\mathcal{A}$ runs in time $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H| + \ell)$.

We sketch the proof of Theorem 4.9, for the simpler case where the code is $\mathrm{RM}[\mathbb{F}, m, d, H]$ (i.e., without partial sums). We can view a polynomial $Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ as a vector over the monomial basis, with an entry for each possible monomial $X_1^{i_1} \ldots X_m^{i_m}$ (with $0 \leq i_1, \ldots, i_m < d$) containing the corresponding coefficient. The evaluation of Q at a point $\vec{\alpha} \in \mathbb{F}^m$ then equals the inner product of this vector with the vector $\phi_{\vec{\alpha}}$, in the same basis, whose entry for $X_1^{i_1} \ldots X_m^{i_m}$ is equal to $\alpha_1^{i_1} \ldots \alpha_m^{i_m}$. Given $\vec{\alpha}_1, \ldots, \vec{\alpha}_\ell$, we could use Gaussian elimination on $\phi_{\vec{\alpha}_1}, \ldots, \phi_{\vec{\alpha}_\ell}$ to check for linear dependencies, which would be equivalent to constraint detection for $\mathrm{RM}[\mathbb{F}, m, d, H]$.

However, we cannot afford to explicitly write down $\phi_{\vec{\alpha}}$, because it has d^m entries. Nevertheless, we can still implicitly check for linear dependencies, and we do so by reducing the problem, by building on and extending ideas of [BW04], to computing the nullspace of a certain set of polynomials, which can be solved via an algorithm of [RS05] (see also [Kay10]). The idea is to encode the entries of these vectors via a succinct description: a polynomial $\Phi_{\vec{\alpha}}$ whose coefficients (after expansion) are the entries of $\phi_{\vec{\alpha}}$. In our setting this polynomial has the particularly natural form:

$$\Phi_{\vec{\alpha}}(\vec{X}) := \prod_{i=1}^{m} (1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1}) \ ;$$

note that the coefficient of each monomial equals its corresponding entry in $\phi_{\vec{\alpha}}$. Given this representation we can use standard polynomial identity testing techniques to find linear dependencies between these polynomials, which corresponds to linear dependencies between the original vectors. Crucially, we cannot afford any mistake, even with exponentially small probability, when looking for linear dependencies for otherwise we would not achieve perfect simulation; this is why the techniques we leverage rely on derandomization. We now proceed with the full proof.

[6] Note that $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$ is indeed linear: for every $w_{Q_1}, w_{Q_2} \in \Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$, $a_1, a_2 \in \mathbb{F}$, and $\vec{\alpha} \in \mathbb{F}^{\leq m}$, it holds that $a_1 w_{Q_1}(\vec{\alpha}) + a_2 w_{Q_2}(\vec{\alpha}) = a_1 \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} Q_1(\vec{\alpha}, \vec{\gamma}) + a_2 \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} Q_2(\vec{\alpha}, \vec{\gamma}) = \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} (a_1 Q_1 + a_2 Q_2)(\vec{\alpha}, \vec{\gamma}) = w_{a_1 Q_1 + a_2 Q_2}(\vec{\alpha})$. But $w_{a_1 Q_1 + a_2 Q_2} \in \Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$, since $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ is a linear space.

Proof of Theorem 4.9. We first introduce some notation. Define $[< d] := \{0, \ldots, d-1\}$. For vectors $\vec{\alpha} \in \mathbb{F}^m$ and $\vec{a} \in [< d]^m$, we define $\vec{\alpha}^{\vec{a}} := \prod_{i=1}^m \alpha_i^{a_i}$; similarly, for variables $\vec{X} = (X_1, \ldots, X_m)$, we define $\vec{X}^{\vec{a}} := \prod_{i=1}^m X_i^{a_i}$.

We identify $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$ with $\mathbb{F}^{[<d]^m}$; a codeword w_Q then corresponds to a vector $\vec{Q}$ whose $\vec{a}$-th entry is the coefficient of the monomial $\vec{X}^{\vec{a}}$ in Q. For $\vec{\alpha} \in \mathbb{F}^{\leq m}$, let

$$\phi_{\vec{\alpha}} := \left(\vec{\alpha}^{\vec{a}} \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \vec{\gamma}^{\vec{b}} \right)_{\vec{a} \in [<d]^{|\vec{\alpha}|}, \vec{b} \in [<d]^{m-|\vec{\alpha}|}} .$$

We can also view $\phi_{\vec{\alpha}}$ as a vector in $\mathbb{F}^{[<d]^m}$ by merging the indices, so that, for all $\vec{\alpha} \in \mathbb{F}^{\leq m}$ and $w_Q \in \Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$,

$$\begin{aligned} w_Q(\vec{\alpha}) &= \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} Q(\vec{\alpha}, \vec{\gamma}) = \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \sum_{\vec{a} \in [<d]^{|\vec{\alpha}|}} \sum_{\vec{b} \in [<d]^{m-|\vec{\alpha}|}} \vec{Q}_{\vec{a},\vec{b}} \cdot \vec{\alpha}^{\vec{a}} \vec{\gamma}^{\vec{b}} \\ &= \sum_{\vec{a} \in [<d]^{|\vec{\alpha}|}} \sum_{\vec{b} \in [<d]^{m-|\vec{\alpha}|}} \vec{Q}_{\vec{a},\vec{b}} \cdot \vec{\alpha}^{\vec{a}} \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \vec{\gamma}^{\vec{b}} = \langle \vec{Q}, \phi_{\vec{\alpha}} \rangle . \end{aligned}$$

Hence for every $\vec{\alpha}_1, \ldots, \vec{\alpha}_\ell, \vec{\alpha} \in \mathbb{F}^{\leq m}$ and $a_1, \ldots, a_\ell \in \mathbb{F}$, the following statements are equivalent (i) $w(\vec{\alpha}) = \sum_{i=1}^\ell a_i w(\vec{\alpha}_i)$ for all $w \in \Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$; (ii) $\langle \vec{f}, \phi_{\vec{\alpha}} \rangle = \sum_{i=1}^\ell a_i \langle \vec{f}, \phi_{\vec{\alpha}_i} \rangle$ for all $\vec{f} \in \mathbb{F}^{[<d]^m}$ (iii) $\phi_{\vec{\alpha}} = \sum_{i=1}^\ell a_i \phi_{\vec{\alpha}_i}$. We deduce that constraint detection for $\Sigma\mathrm{RM}[\mathbb{F}, m, d, H]$ is equivalent to the problem of finding $a_1, \ldots, a_\ell \in \mathbb{F}$ such that $\phi_{\vec{\alpha}} = \sum_{i=1}^\ell a_i \phi_{\vec{\alpha}_i}$, or returning 'independent' if no such $a_1, \ldots, a_\ell$ exist.

However, the dimension of the latter vectors is d^m, which may be much larger than $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H| + \ell)$, and so we cannot afford to "explicitly" solve the $\ell \times d^m$ linear system. Instead, we "succinctly" solve it, by taking advantage of the special structure of the vectors, as we now describe. For $\vec{\alpha} \in \mathbb{F}^m$, define the polynomial

$$\Phi_{\vec{\alpha}}(\vec{X}) := \prod_{i=1}^m (1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1}) .$$

Note that, while the above polynomial is computable via a small arithmetic circuit, its coefficients (once expanded over the monomial basis) correspond to the entries of the vector $\phi_{\vec{\alpha}}$. More generally, for $\vec{\alpha} \in \mathbb{F}^{\leq m}$, we define the polynomial

$$\Phi_{\vec{\alpha}}(\vec{X}) := \left(\prod_{i=1}^{|\vec{\alpha}|} (1 + \alpha_i X_i + \cdots + \alpha_i^{d-1} X_i^{d-1}) \right) \left(\prod_{i=1}^{m-|\vec{\alpha}|} \sum_{\gamma \in H} (1 + \gamma X_{i+|\vec{\alpha}|} + \cdots + \gamma^{d-1} X_{i+|\vec{\alpha}|}^{d-1}) \right) .$$

Note that $\Phi_{\vec{\alpha}}$ is a product of univariate polynomials. To see that the above does indeed represent $\phi_{\vec{\alpha}}$, we rearrange the expression as follows:

$$\begin{aligned} \Phi_{\vec{\alpha}}(\vec{X}) &= \left(\prod_{i=1}^{|\vec{\alpha}|} (1 + \alpha_i X_i + \cdots + \alpha_i^{d-1} X_i^{d-1}) \right) \left(\sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \prod_{i=1}^{m-|\vec{\alpha}|} (1 + \gamma_i X_{i+|\vec{\alpha}|} + \cdots + \gamma_i^{d-1} X_{i+|\vec{\alpha}|}^{d-1}) \right) \\ &= \Phi_{\vec{\alpha}}(X_1, \ldots, X_{|\vec{\alpha}|}) \left(\sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \Phi_{\vec{\gamma}}(X_{|\vec{\alpha}|+1}, \ldots, X_m) \right) ; \end{aligned}$$

indeed, the coefficient of $\vec{X}^{\vec{a},\vec{b}}$ for $\vec{a} \in [< d]^{|\vec{\alpha}|}$ and $\vec{b} \in [< d]^{m-|\vec{\alpha}|}$ is $\vec{\alpha}^{\vec{a}} \sum_{\vec{\gamma} \in H^{m-|\vec{\alpha}|}} \vec{\gamma}^{\vec{b}}$, as required.

Thus, to determine whether $\phi_\alpha \in \mathrm{span}(\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell})$, it suffices to determine whether $\Phi_\alpha \in \mathrm{span}(\Phi_{\alpha_1}, \ldots, \Phi_{\alpha_\ell})$. In fact, the linear dependencies are in correspondence: for $a_1, \ldots, a_\ell \in \mathbb{F}$, $\phi_\alpha = \sum_{i=1}^\ell a_i \phi_{\alpha_i}$ if and only if $\Phi_\alpha = \sum_{i=1}^\ell a_i \Phi_{\alpha_i}$. Crucially, each Φ_{α_i} is not only in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ but is a product of m univariate polynomials each represented via an $\mathbb{F}$-arithmetic circuit of size $\mathrm{poly}(|H| + d)$. We leverage this special structure and solve the above problem by relying on an algorithm of [RS05] that computes the nullspace for such polynomials (see also [Kay10]), as captured by the lemma below;[7] for completeness, we provide an elementary proof of the lemma in Appendix D.

[7] One could use polynomial identity testing to solve the above problem in probabilistic polynomial time; see [Kay10, Lemma 8]. However, due to a nonzero probability of error, this suffices only to achieve statistical zero knowledge, but *does not suffice to achieve perfect zero knowledge*.

Lemma 4.11. *There exists a deterministic algorithm $\mathcal{D}$ such that, on input a vector of m-variate polynomials $\vec{Q} = (Q_1, \ldots, Q_\ell)$ over $\mathbb{F}$ where each polynomial has the form $Q_k(\vec{X}) = \prod_{i=1}^{m} Q_{k,i}(X_i)$ and each $Q_{k,i}$ is univariate of degree less than d with $d \leq |\mathbb{F}|$ and represented via an $\mathbb{F}$-arithmetic circuit of size s, outputs a basis for the linear space $\vec{Q}^{\perp} := \{(a_1, \ldots, a_\ell) \in \mathbb{F}^\ell : \sum_{k=1}^{\ell} a_k Q_k \equiv 0\}$. Moreover, $\mathcal{D}$ runs in $\mathrm{poly}(\log |\mathbb{F}| + m + d + s + \ell)$ time.*

The above lemma immediately provides a way to construct a constraint detector for $\Sigma\mathrm{RM}$: given as input an index $\mathfrak{n} = (\mathbb{F}, m, d, H)$ and a subset $I \subseteq D(\mathfrak{n})$, we construct the arithmetic circuit Φ_α for each $\alpha \in I$, and then run the algorithm $\mathcal{D}$ on vector of circuits $(\Phi_\alpha)_{\alpha \in I}$, and directly output $\mathcal{D}$'s result. The lemma follows. □

4.3 Univariate polynomials with BS proximity proofs

We show that evaluations of univariate polynomials concatenated with corresponding BS proximity proofs [BS08] have succinct constraint detection (see Definition 4.2). Recall that the Reed–Solomon code (see Section 3.4) is not locally testable, but one can test proximity to it with the aid of the quasilinear-size proximity proofs of Ben-Sasson and Sudan [BS08]. These latter apply when low-degree univariate polynomials are evaluated over *linear spaces*, so from now on we restrict our attention to Reed–Solomon codes of this form. More precisely, we consider Reed–Solomon codes $\mathrm{RS}[\mathbb{F}, L, d]$ where $\mathbb{F}$ is an extension field of a base field $\mathbb{K}$, L is a $\mathbb{K}$-linear subspace in $\mathbb{F}$, and $d = |L| \cdot |\mathbb{K}|^{-\mu}$ for some $\mu \in \mathbb{N}^+$. We then denote by $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ the code obtained by concatenating codewords in $\mathrm{RS}[\mathbb{F}, L, |L| \cdot |\mathbb{K}|^{-\mu}]$ with corresponding BS proximity proofs whose recursion terminates at "base dimension" $k \in \{1, \ldots, \dim(L)\}$ (for completeness we include a formal definition of these in Appendix F); typically $\mathbb{K}, \mu, k$ are fixed to certain constants (e.g., [BS08] fixes them to $\mathbb{F}_2, 3, 1$, respectively) but below we state the cost of constraint detection in full generality. The linear code family BS-RS is indexed by tuples $\mathfrak{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$ and the $\mathfrak{n}$-th code is $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$, and our result about BS-RS is the following:

Theorem 4.12 (formal statement of 1.7)**.** BS-RS *has* $\mathrm{poly}(\log |\mathbb{F}| + \dim(L) + |\mathbb{K}|^\mu + \ell)$*-time constraint detection.*

The proof of the above theorem is technically involved, and we present it via several steps, as follows. (1) In Section 4.3.1 we introduce the notion of a *code cover* and two key combinatorial properties of these: κ*-locality* and κ*-independence*. (2) In Section 4.3.2 we introduce the notion of a *recursive* code cover and relate its combinatorial properties to those of (standard) code covers. (3) In Section 4.3.3 we show how to construct succinct constraint detectors starting from algorithms that detect constraints only 'locally' for code covers and recursive code covers. (4) In Section 4.3.4 we show that BS-RS has a recursive code cover with the requisite properties and thus implies, via the results of prior steps, a succinct constraint detector, as claimed. Several sub-proofs are deferred to the appendices, and we provide pointers to these along the way.

The role of code covers. We are interested in succinct constraint detection: solving the constraint detection problem for certain code families with exponentially-large domains (such as BS-RS). We now build some intuition about how code covers can, in some cases, facilitate this.

Consider the simple case where the code $C \subseteq \mathbb{F}^D$ is a direct sum of many small codes: there exists $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ such that $D = \cup_j \tilde{D}_j$ and $C = \oplus_j \tilde{C}_j$ where, for each j, $\tilde{C}_j$ is a linear code in $\mathbb{F}^{\tilde{D}_j}$ and the subdomain $\tilde{D}_j$ is small and disjoint from other subdomains. The detection problem for this case can be solved efficiently: use the generic approach of Gaussian elimination independently on each subdomain $\tilde{D}_j$.

Next consider a more general case where the subdomains are not necessarily disjoint: there exists $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ as above but we do not require that the $\tilde{D}_j$ form a partition of D; we say that each $(\tilde{D}_j, \tilde{C}_j)$ is a *local view* of C because $\tilde{D}_j \subseteq D$ and $\tilde{C}_j = C|_{\tilde{D}_j}$, and we say that S is a *code cover* of C. Now suppose that for each j there exists an efficient constraint detector for $\tilde{C}_j$ (which is defined on $\tilde{D}_j$); in this case, the detection problem can be solved efficiently at least for those subsets I that are contained in $\tilde{D}_j$ for some j. Generalizing further, we see that we can efficiently solve constraint detection for a code C if there is a cover $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ such that, given a subset $I \subseteq D$, (i) I is contained in some subdomain $\tilde{D}_j$, and (ii) constraint detection for $\tilde{C}_j$ can be solved efficiently.

We build on the above ideas to derive analogous statements for recursive code covers, which arise naturally in the case of BS-RS. But note that recursive constructions are common in the PCP literature, and we believe that our cover-based techniques are of independent interest as, e.g., they are applicable to *other* PCPs, including [BFLS91, AS98].

4.3.1 Covering codes with local views

The purpose of this section is to formally define the notion of cover and certain combinatorial properties of these.

Definition 4.13. *Let C be a linear code with domain D and alphabet $\mathbb{F}$. A* **(local) view** *of C is a pair $(\tilde{D}, \tilde{C})$ such that $\tilde{D} \subseteq D$ and $C|_{\tilde{D}} = \tilde{C}$. A* **cover** *of C is a set of local views $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ of C such that $D = \cup_j \tilde{D}_j$. Also, we define a cover's domain intersection as $\mathrm{di}(S) := \cup_{i \neq j}(\tilde{D}_i \cap \tilde{D}_j)$ and, given a set J, we define $\tilde{D}_J := \cup_{j \in J} \tilde{D}_j$.*

Example 4.14 (line cover of RM)**.** Suppose for instance that C is the Reed–Muller code $\mathrm{RM}[\mathbb{F}, \mathbb{F}, m, d]$: C consists of evaluations over $D = \mathbb{F}^m$ of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ (see Definition 3.4). A cover of C that is extensively studied in the PCP and property-testing literature is the one given by (axis-parallel) *lines*. A line (in the i-th direction) is a set of $|\mathbb{F}|$ points that agree on all but one coordinate (the i-th one); and the *line cover* of C is thus $S = \{(\tilde{D}_\ell, \tilde{C}_\ell)\}$ where ℓ ranges over all (axis-parallel) lines and $\tilde{C}_\ell$ is the Reed–Solomon code $\mathrm{RS}[\mathbb{F}, \mathbb{F}, d]$ (see Definition 3.4).

Observe that the domain intersection of the line cover equals $\mathbb{F}^m$, which is also the domain D of the base code C. However, for BS-RS, we consider a cover whose domain intersection is a strict subset of D (see Appendix G).

Next, we specify a notion of *locality* for covers. A partial assignment $w' \in \mathbb{F}^{D'}$ is *locally consistent* with a cover $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ if for every local view $(\tilde{D}_j, \tilde{C}_j)$ with $\tilde{D}_j \subseteq D'$ the restriction $w'|_{\tilde{D}_j}$ is a codeword of $\tilde{C}_j$. Then we say that a cover is *κ-local* if any locally consistent assignment $w' \in \mathbb{F}^{D'}$, where D' is a union of at most κ domains in the cover, can be extended to a "globally consistent" codeword w of C.

Definition 4.15. *Let C be a linear code with domain D and alphabet $\mathbb{F}$. Given $\kappa \in \mathbb{N}$, a* **κ-local cover** *of C is a cover $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ of C such that: for every subset of view-indices J of size at most κ and every word $w' \in \mathbb{F}^{\tilde{D}_J}$ with $w'|_{\tilde{D}_j} \in \tilde{C}_j$ (for every $j \in J$), the word w' can be extended to some word w in C, i.e., w satisfies $w|_{\tilde{D}_J} = w'$). (The trivial cover $S = \{(D, C)\}$ of C is κ-local for every κ.)*

The following definition significantly strengthens the previous one. Informally, a cover is *κ-independent* if every partial assignment over a subdomain D' that is the union of κ subdomains from the cover *and* κ auxiliary locations can be extended to a "globally consistent" codeword. We use this stronger notion in our main Lemma 4.20.

Definition 4.16. *Let C be a linear code with domain D and alphabet $\mathbb{F}$. Given $\kappa \in \mathbb{N}$, a* **κ-independent cover** *of C is a κ-local cover $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ such that for every set J of size at most κ, subdomain $D' \subseteq \mathrm{di}(S)$ of size at most κ, and $w' \in \mathbb{F}^{D' \cup \tilde{D}_J}$ with $w'|_{\tilde{D}_j} \in \tilde{C}_j$ for every $j \in J$, there exists $w \in C$ such that $w|_{D' \cup \tilde{D}_J} = w'$. (The trivial cover $S = \{(D, C)\}$ of C is κ-independent for every $\kappa \in \mathbb{N}$ because it is κ-local and has $\mathrm{di}(S) = \emptyset$.)*

Example 4.17. The line cover from Example 4.14 is d-local because any evaluation on at most d (axis-parallel) lines $\ell_1, \ldots, \ell_d$ that is a polynomial of degree $(d-1)$ along each of $\ell_1, \ldots, \ell_d$ can be extended to a codeword of $\mathrm{RM}[\mathbb{F}, \mathbb{F}, m, d]$. Furthermore, it is $d/2$-independent because any locally consistent assignment to $d/2$ such lines and $d/2$ points $p_1, \ldots, p_{d/2}$ can be extended to a valid Reed–Muller codeword. To see this, observe that there exists an interpolating set $H_1 \times \cdots \times H_m$ (with $|H_i| = d$) that contains $p_1, \ldots, p_{d/2}$ and intersects each line in d points; we shall later use this observation for the bivariate case ($m = 2$), a proof for that case is provided at Claim H.1.

4.3.2 Recursive covers and locality

Proximity proofs for codes such as the Reed–Solomon code and the Reed–Muller code are typically obtained via techniques of proof composition [AS98]. Informally, a problem is reduced to a set of smaller sub-problems of the same kind (which are usually interconnected), and a sub-proof is constructed for each sub-problem. This process leads to a proof for the original problem that is "covered" by the sub-proofs for the sub-problems, and naturally imply a cover of the proof by these sub-proofs. This process is then repeated recursively until the sub-problems are small enough for the verifier to check directly — and in our case leads to the notion of *recursive covers*, which we define below.

To support the definition of a recursive cover, we first introduce notation for rooted trees. Edges in a rooted tree $T = (V, E)$ are directed from the root r towards the leaves; the edge directed from v to u is denoted (v, u); v is the *predecessor* of u and u the *successor* of v; if there is a path from v to v' we say that v is an *ancestor* of v'; if there is no directed path between v and v' (in either direction) we say that the two vertices are *disconnected*. The *set of successors*

of v is denoted $\text{successors}(T, v)$. The *depth* of a vertex v in T is denoted $\text{depth}(T, v)$ and equals the number of edges on the path from r to v. The depth of T is denoted $\text{depth}(T)$ and equals the maximum of $\text{depth}(T, v)$ as v ranges in V. The i-th *layer* of T is denoted $\text{layer}(T, i)$ and equals the set of $v \in V$ such that $\text{depth}(T, v) = i$. (Note that $\text{depth}(T, r) = 0$ and $\text{layer}(T, 0) = \{r\}$.) An *equidepth* tree is a tree in which all leaves have equal depth.

Definition 4.18. *Let C be a linear code with domain D and alphabet $\mathbb{F}$. A* **recursive cover** *of C is a directed rooted equidepth tree T of non-zero depth where each vertex v is labeled by a view $(\tilde{C}_v, \tilde{D}_v)$ such that: (i) $\tilde{C}_v$ is a linear code with domain $\tilde{D}_v$ and alphabet $\mathbb{F}$; (ii) if v is the root, then $(\tilde{C}_v, \tilde{D}_v) = (C, D)$; and (iii) for every non-leaf v the set $T_v := \{(\tilde{C}_u, \tilde{D}_u)\}_{u \in \text{successors}(T,v)}$ is a cover of $\tilde{C}_v$. Furthermore we define the following notions:*

- *Given $d \in \{0, \ldots, \text{depth}(T)\}$, the d-***depth restriction** *of T is $T|_d := \bigcup_{v \in \text{layer}(T,d)} \{(\tilde{C}_v, \tilde{D}_v)\}$. (Note that $T|_0 = \{(C, D)\}$.)*
- *Given $c \in \mathbb{N}$, we say that T is c-***intersecting** *if $|\tilde{D}_u \cap \tilde{D}_{u'}| \leq c$ for every two disconnected vertices u, v.*
- *Given $\kappa \in \mathbb{N}$, we say that T is κ-***independent** *if T_v is a κ-independent cover of $\tilde{C}_v$ for every non-leaf vertex v in T.*

Remark 4.19. The above definition is restricted to equidepth trees, but can be extended to general trees as follows. Iteratively append to each leaf v of non-maximal depth a single successor u labeled by $(\tilde{C}_u, \tilde{D}_u) := (\tilde{C}_v, \tilde{D}_v)$; this leads to a cover of T_v that is 0-intersecting and κ-doubly independent for κ that equals $\tilde{C}_v$'s dual distance.

Below we state the main lemma of this section. This lemma says that (given certain restrictions) if a recursive cover has the *local* property of *independence* (of some degree) at each internal vertex, then each of its layers has the *global* property of *locality* (of some degree) as a cover of the root. Later on (in Section 4.3.3) we show how cover locality is used to construct constraint detectors.

Lemma 4.20 (main)**.** *Let C be a linear code with domain D and alphabet $\mathbb{F}$, and let T be a recursive cover of C such that (i) T is c-intersecting for $c > 0$, and (ii) for every non-leaf vertex v in T it holds that T_v is a κ-independent cover of $\tilde{C}_v$. Then, for every $d \in \{0, \ldots, \text{depth}(T)\}$, $T|_d$ is a $\frac{\kappa}{c}$-local cover of C.*

Proof. We prove the statement by induction on the non-negative integer d. The base case is when $d = 0$, and holds because $T|_0 = \{(D, C)\}$ is the trivial cover, thus it is a κ'-local cover of C for any $\kappa' \geq 0$ and, in particular, a $\frac{\kappa}{c}$-local cover. We now assume the statement for $d < \text{depth}(T)$ and prove it for depth $d + 1$.

Let $T|_d = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ be the d-depth cover of C, and let $T|_{d+1} = \{(\tilde{D}_{i,j}, \tilde{C}_{i,j})\}_{i,j}$ be the $(d+1)$-depth cover of C, where, for every i, $T|^{(i)}_{d+1} = \{(\tilde{D}_{i,j}, \tilde{C}_{i,j})\}_j$ is the cover of $\tilde{C}_i$ (this can be ensured via suitable indexing). Let J be a set of pairs (i, j) of size at most $\frac{\kappa}{c}$, and let $w' \in \mathbb{F}^{\tilde{D}_J}$ be such that $w'|_{\tilde{D}_{i,j}} \in \tilde{C}_{i,j}$ for every $(i, j) \in J$. We show that there exists $w \in C$ such that $w|_{\tilde{D}_J} = w'$. Define $I := \{i : \exists\, j \text{ s.t. } (i, j) \in J\}$ and note that $|I| \leq |J| \leq \frac{\kappa}{c}$. By the inductive assumption, it suffices to show that there exists $w \in \mathbb{F}^{\tilde{D}_I}$ such that (a) $w|_{\tilde{D}_{i,j}} = w'|_{\tilde{D}_{i,j}}$ for every $(i, j) \in J$, and (b) $w|_{\tilde{D}_i} \in \tilde{C}_i$ for every $i \in I$.

For simplicity assume $I = \{1, \ldots, |I|\}$. We construct w incrementally and view w as belonging to $(\mathbb{F} \cup \{\emptyset\})^{\tilde{D}_I}$, i.e., it is a partial mapping from $\tilde{D}_I$ to $\mathbb{F}$. Let $\text{def}(w) := \{\alpha \in \tilde{D}_I : w(\alpha) \neq \emptyset\}$ denote the set of locations where w is defined. Initialize $\text{def}(w) = \tilde{D}_J$ and $w|_{\tilde{D}_J} = w'$; then, for increasing $i = 1, \ldots, |I|$, iteratively extend w to be defined (also) over $\tilde{D}_i$, eventually obtaining $w \in \mathbb{F}^{\tilde{D}_I}$. In the i-th iteration (that handles $\tilde{D}_i$), it is sufficient to prove the existence of a codeword $w_i \in \tilde{C}_i$ such that $w_i|_{\tilde{D}_i \cap \text{def}(w)} = w|_{\tilde{D}_i \cap \text{def}(w)}$. If such a codeword exists then we shall define w on $\tilde{D}_i$ by $w|_{\tilde{D}_i} = w_i$, thus eventually reaching w that satisfies the stated requirements.

To show that during the i-th iteration the desired w_i exists, partition the elements of $\tilde{D}_i \cap \text{def}(w)$ into two sets: $V := \cup_{j \text{ s.t. } (i,j) \in J} \tilde{D}_{i,j}$ and $W := (\tilde{D}_i \cap \text{def}(w)) \setminus V$. Note that $W \subseteq \cup_{i' \neq i} (\tilde{D}_i \cap \tilde{D}_{i'})$, because defining $w(\alpha)$ for any $\alpha \in W$ can be done only: (i) in the initialization phase, so that $\alpha \in \tilde{D}_{i',j'} \subseteq \tilde{D}_{i'}$ for some $(i', j') \in J$ with $i' \neq i$ (as otherwise $\alpha \in V$); or (ii) in a previous iteration, so that $\alpha \in \tilde{D}_{i'}$ for some $i' < i$ (as $\tilde{D}_{i'}$ was already handled and w is already defined on all of $\tilde{D}_{i'}$). The above implies that $W \subseteq \text{di}(T|^{(i)}_{d+1})$ and the assumption that T is c-intersecting implies

$$|W| \leq \sum_{i' \neq i} |\tilde{D}_i \cap \tilde{D}_{i'}| \leq c \cdot |I| \leq c \cdot |J| \leq \kappa \ .$$

Similarly, note that for every fixed $i \in I$ the number of pairs $(i, j) \in J$ is at most $|J| \leq \kappa$. By assumption $\tilde{C}_i$ has a κ-independent cover and thus we conclude (via Definition 4.16) that the desired w_i exists, as required. □

4.3.3 From recursive covers to succinct constraint detection

The purpose of this section is to establish sufficient conditions for succinct constraint detection by leveraging covers with small-enough views and large-enough locality. First, in Definition 4.21 and Lemma 4.22, we define *cover-based constraint detection* and prove that it implies succinct constraint detection; informally, we consider the case when a code has a sequence of covers where view size and locality reduce together, and prove that we can locally detect constraints in a number of views that is proportional to the constraint's weight and each view's size is proportional to the constraint's weight, by choosing the right cover from the sequence. Then, in Definition 4.24 and Lemma 4.25, we extend our discussion to recursive code covers by defining *recursive-cover-based constraint detection* and establishing that it implies the previous notion. We conclude (in Corollary 4.26) that recursive-cover-based constraint detection implies succinct constraint detection.

Definition 4.21. *Let $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. We say that $\mathscr{C}$ has* **cover-based constraint detection** *if there exists an algorithm that, given an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, outputs in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time a subset $W \subseteq \mathbb{F}(\mathfrak{n})^{D(\mathfrak{n})}$ for which there exists a subset S' of some $|I|$-local cover S of $C_\mathfrak{n}$, and the following holds: (i) $|S'| \leq |I|$; (ii) $I \subseteq (\cup_{(\tilde{D},\tilde{C})\in S'} \tilde{D})$; (iii) $\mathrm{span}(W) = \mathrm{span}(\cup_{(\tilde{D},\tilde{C})\in S'} \tilde{C}^\perp)$.*

Lemma 4.22. *Let $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. If $\mathscr{C}$ has cover-based constraint detection then $\mathscr{C}$ has succinct constraint detection.*

To prove this lemma we require a technical claim, the proof of which is deferred to Appendix E.

Claim 4.23. *Let C be a linear code with domain D and alphabet $\mathbb{F}$, let $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ be a κ-local cover of C. For any set J of size at most κ it holds $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^\perp) = (C^\perp)_{\subseteq\left(\cup_{j\in J}\tilde{D}_j\right)}$.*

Proof of Lemma 4.22. By Lemma 4.6, it suffices to show an algorithm that, on input an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, outputs a subset $W \subseteq \mathbb{F}(\mathfrak{n})^{D(\mathfrak{n})}$ with $(C_\mathfrak{n}^\perp)_{\subseteq I} \subseteq \mathrm{span}(W) \subseteq C_\mathfrak{n}^\perp$ in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time. We take this algorithm to be the one guaranteed by Definition 4.21. To see correctness, let $\tilde{D}_{S'} := \cup_{(\tilde{D},\tilde{C})\in S'} \tilde{D}$, and note that Definition 4.21 and Claim 4.23 imply that $\mathrm{span}(W) = (C_\mathfrak{n}^\perp)_{\subseteq \tilde{D}_{S'}}$ and $(C_\mathfrak{n}^\perp)_{\subseteq I} \subseteq (C_\mathfrak{n}^\perp)_{\subseteq \tilde{D}_{S'}} \subseteq C_\mathfrak{n}^\perp$, as required. □

Next we show that, under certain conditions, code families with recursive covers imply a sequence of covers that we can use to construct cover-based constraint detectors. Combined with Lemma 4.22, this result is key for establishing a connection from certain proximity proof constructions to succinct constraint detectors.

Definition 4.24. *Let $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. We say that $\mathscr{C}$ has* **recursive-cover-based constraint detection** *if:*

- *there exists $c \in \mathbb{N}$ such that, for every index $\mathfrak{n}$, $C_\mathfrak{n}$ has a c-intersecting recursive cover $T_\mathfrak{n}$;*
- *there exists an algorithm that, given an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, outputs in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time a subset $W \subseteq \mathbb{F}(\mathfrak{n})^{D(\mathfrak{n})}$ for which there exist $d \in \{0, \ldots, \mathrm{depth}(T_\mathfrak{n})\}$ and $U \subseteq \mathrm{layer}(T_\mathfrak{n}, d)$ such that: (i) for every vertex v in $T_\mathfrak{n}$ with $\mathrm{depth}(T_\mathfrak{n}, v) < d$, the cover $T_{\mathfrak{n},v}$ is $c|I|$-independent; (ii) $|U| \leq |I|$; (iii) $I \subseteq (\cup_{u\in U}\tilde{D}_u)$; (iv) $\mathrm{span}(W) = \mathrm{span}(\cup_{u\in U}\tilde{C}_u^\perp)$.*

Lemma 4.25. *Let $\mathscr{C} = \{C_\mathfrak{n}\}_\mathfrak{n}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. If $\mathscr{C}$ has recursive-cover-based constraint detection, then $\mathscr{C}$ has cover-based constraint detection.*

Proof. The definition of recursive-cover-based detection says that there exist (a) $c \in \mathbb{N}$ such that, for every index $\mathfrak{n}$, $C_\mathfrak{n}$ has a c-intersecting recursive cover $T_\mathfrak{n}$, and (b) an algorithm satisfying certain properties. We show that this algorithm meets the requirements for being a cover-based constraint detector (see Definition 4.21). Consider any index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, and let W be the output of the algorithm. Let $d \in \{0, \ldots, \mathrm{depth}(T_\mathfrak{n})\}$ and $U \subseteq \mathrm{layer}(T_\mathfrak{n}, d)$ be the objects associated to W (guaranteed by the definition of recursive-cover-based constraint detection). Let $S := T_\mathfrak{n}|_d$

(i.e., S is the d-depth restriction of $T_{\mathfrak{n}}$) and $S' := \{(\tilde{D}_u, \tilde{C}_u)\}_{u \in U}$; it suffices to show that S is $|I|$-local. The claim follows directly by the assumption on d and Lemma 4.20, because $T_{\mathfrak{n},v}$ is $c|I|$-independent for every vertex v in $T_{\mathfrak{n}}$ with $\mathrm{depth}(T_{\mathfrak{n}}, v) < d$, and thus $S = T_{\mathfrak{n}}|_d$ is indeed a $|I|$-local cover of C. □

Corollary 4.26. *Let $\mathscr{C} = \{C_{\mathfrak{n}}\}_{\mathfrak{n}}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. If $\mathscr{C}$ has recursive-cover-based constraint detection, then $\mathscr{C}$ has succinct constraint detection.*

Proof. Follows directly from Lemma 4.25 (recursive-cover-based constraint detection implies cover-based constraint detection) and Lemma 4.22 (cover-based constraint detection implies succinct constraint detection). □

4.3.4 Proof of Theorem 4.12

The purpose of this section is to prove Theorem 4.12. By Corollary 4.26, it suffices to argue that the linear code family BS-RS has recursive-cover-based constraint detection (see Definition 4.24).

Recall that we consider Reed–Solomon codes $\mathrm{RS}[\mathbb{F}, L, d]$ where $\mathbb{F}$ is an extension field of a base field $\mathbb{K}$, L is a $\mathbb{K}$-linear subspace in $\mathbb{F}$, and $d = |L| \cdot |\mathbb{K}|^{-\mu}$ for some $\mu \in \mathbb{N}$; and we denote by $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ the code obtained by concatenating codewords in $\mathrm{RS}[\mathbb{F}, L, |L| \cdot |\mathbb{K}|^{-\mu}]$ with corresponding [BS08] proximity proofs with "base dimension" $k \in \{1, \ldots, \dim(L)\}$ (see Appendix F for details). The linear code family BS-RS is indexed by tuples $\mathfrak{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$ and the $\mathfrak{n}$-th code is $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$.

We represent indices $\mathfrak{n}$ so that $\log|\mathbb{F}| + \dim(L) + |\mathbb{K}|^{\mu} \leq \mathrm{poly}(|\mathfrak{n}|)$. The base field $\mathbb{K}$ and extension field $\mathbb{F}$ require $O(\log|\mathbb{K}|)$ and $O(\log|\mathbb{F}|)$ bits to represent; the subspace L requires $O(\dim(L))$ elements in $\mathbb{F}$ to represent; and the two integers μ and k require $O(\log\mu)$ and $O(\log k)$ bits to represent. In addition, we add $|\mathbb{K}|^{\mu}$ arbitrary bits of padding. Overall, we obtain that $|\mathfrak{n}| = \Theta(\log|\mathbb{K}| + \log|\mathbb{F}| + \log|\mathbb{F}| \cdot \dim(L) + \log\mu + \log k + |\mathbb{K}|^{\mu}) = \Theta(\log|\mathbb{F}| \cdot \dim(L) + |\mathbb{K}|^{\mu})$.

The main claim in this section is the following (and does not rely on fixing $\mathbb{K}, \mu$).

Lemma 4.27. *Define the depth function $d(\mathbb{K}, L, \mu, a) := \log_2 \dim(L) - \log_2(\log_{|\mathbb{K}|} a + \mu + 2) - 1$. The linear code family* BS-RS *satisfies the following properties.*

- *For every index $\mathfrak{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$, $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ has a 1-intersecting recursive cover $T_{\mathfrak{n}}$. Also, for every positive integer m and non-leaf vertex v in $T_{\mathfrak{n}}$ with $\mathrm{depth}(T_{\mathfrak{n}}, v) < d(\mathbb{K}, L, \mu, m)$, the cover $T_{\mathfrak{n},v}$ is m-independent.*
- *There exists an algorithm that, given an index $\mathfrak{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$ and subset $I \subseteq D(\mathfrak{n})$, outputs in time $\mathrm{poly}(\log|\mathbb{F}| + \dim(L) + |\mathbb{K}|^{\mu} + |I|)$ a subset $W \subseteq \mathbb{F}^{D(\mathfrak{n})}$ for which there exist $U \subseteq \mathrm{layer}(T_{\mathfrak{n}}, d(\mathbb{K}, L, \mu, |I|))$ such that: (i) $|U| \leq |I|$; (ii) $I \subseteq (\cup_{u \in U} \tilde{D}_u)$; (iii) $\mathrm{span}(W) = \mathrm{span}(\cup_{u \in U} \tilde{C}_u^{\perp})$.*

Given the above lemma, we can complete the proof of Theorem 4.12, as explained below. We defer the (long and technical) proof of the lemma to Appendix G, and instead end this section with an overview of that proof.

Proof of Theorem 4.12. The proof follows from Lemma 4.27 above and from Corollary 4.26, as we now explain.

Corollary 4.26 states that if a linear code family $\mathscr{C}$ has recursive-cover-based constraint detection (see Definition 4.24), then $\mathscr{C}$ has succinct constraint detection (see Definition 4.2). Also recall that the definition of recursive-cover-based detection requires having a c-intersecting recursive cover for each code in the class, and an algorithm satisfying certain properties.

Observe that Lemma 4.27 guarantees that every code in BS-RS has a 1-intersecting recursive code and, moreover, guarantees the existence of an algorithm whose output satisfies the required properties. We are left to argue that the algorithm runs in time $\mathrm{poly}(|\mathfrak{n}| + |I|)$. But this immediately follows from the running time stated in Lemma 4.27 and the fact that $\log|\mathbb{F}| + \dim(L) + |\mathbb{K}|^{\mu} \leq \mathrm{poly}(|\mathfrak{n}|)$. □

Overview of Lemma 4.27's proof. We assume familiarity with the linear code family BS-RS from [BS08]; for completeness, we provide formal definitions and notations in Appendix F. Recall that the Reed–Solomon code is not locally testable, but one can test proximity to it with the aid of BS proximity proofs [BS08]; the linear code family BS-RS consists of the concatenation of Reed–Solomon codes with BS corresponding proximity proofs.

The construction of the aforementioned proximity proofs is *recursive*, with each step in the recursion reducing both the evaluation domain size $|L|$ and the degree d to (approximately) their square roots. Namely, testing proximity of a

codeword w to $\mathrm{RS}[\mathbb{F}, L, d]$ is reduced to testing proximity of $\Theta(\sqrt{|L|})$ codewords $\{w_i\}_i$ to $\{\mathrm{RS}[\mathbb{F}, L_i, d_i]\}_i$, where $|L_i|, d_i = \Theta(\sqrt{|L|})$ for each i. This step is then recursively applied (by way of proof composition [AS98]) to each codeword w_i, until the domain size is "small enough".

The first part of the proof of Lemma 4.27 consists of various combinatorial claims (see Appendix G.1). First, we observe that the union of the domains of the codewords w_i covers (and, actually, slightly expands) the domain of the original codeword w; this holds recursively, and induces a recursive cover T (see Definition G.3). We prove that T is 1-intersecting (see Claim G.4) and that, for every vertex v in T of depth at most d, the cover T_v is $(|L|^{2^{-d-1}} \cdot |\mathbb{K}|^{-\mu-2})$-independent, which implies the stated independence property about T_v (see Claim G.5). The core of the argument for this second claim is to show that the code $\tilde{C}_v$ equals $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L_v, \mu, k]$ for some subspace L_v such that $\dim(L) \cdot 2^{-d} \leq \dim(\tilde{L}) \leq \dim(L) \cdot 2^{-d} + 2\mu$ (see Claim G.6).

The second part of the proof of Lemma 4.27 consists of establishing the computational efficiency of certain tasks related to the recursive cover (see Appendix G.2). Specifically, we bound the time required to compute a spanning set for covers in T (see Claim G.8). After a few more observations, we are able to conclude the proof.

5 Sumcheck with perfect zero knowledge

We obtain an IPCPP for sumcheck that is perfect zero knowledge against unbounded queries. (Since the input F is an oracle given to the verifier, the proof system is formally an *exact IPCP of proximity for a promise relation*.)

Sumcheck. The sumcheck protocol [LFKN92, Sha92] is an IP for the claim "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$", where F is a polynomial in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ and H is a subset of $\mathbb{F}$. The prover and verifier have input $(\mathbb{F}, m, d, H)$ and oracle access to (the evaluation table on $\mathbb{F}^m$ of) F. The sumcheck protocol has soundness error $1 - (1 - \frac{d}{|\mathbb{F}|})^m$; the prover runs in space $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H|)$ and the verifier in time $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H|)$; the number of rounds is m; finally, the protocol is public coin and the verifier queries F only at one random point.

Leakage. The sumcheck protocol is *not* zero knowledge: a verifier, by interacting with the honest prover, learns partial sums of F, in addition to the fact that "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$" is true. Assuming one way functions, one *can* make any interactive proof, including the sumcheck protocol, to be (computational) zero knowledge [GMR89, IY87, BGG$^+$88]; moreover, one-way functions are necessary for obtaining zero knowledge IPs for non-trivial languages [OW93]. As we do not wish to make intractability assumptions, we now turn to a different proof system model.

Perfect zero knowledge via IPCPPs. We obtain an IPCPP for sumcheck that is perfect zero knowledge against unbounded queries. Namely, a malicious verifier has oracle access to a proof string π and also interacts with the prover, but learns no information about F beyond the fact that the statement about F is true, in the following sense. There exists an algorithm that perfectly simulates the verifier's view by making as many queries to F as the *total* number of verifier queries to either F or the oracle π. (Analogously to zero knowledge for proximity testers, a verifier may query F at any time, so any such information comes "for free" and, also, any query to π 'counts' as a query to F; see Section 3.3.)

Our construction proceeds in two steps:

- *Step 1.* We modify the sumcheck protocol to make it perfect zero knowledge, but in a hybrid model where the prover and verifier have access to a random polynomial $R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$. Crucially, soundness relies only on the fact that R is low-degree, but not the fact that it is random. Also, the modified protocol does *not* depend on a bound on the malicious verifier's queries, and thus maintains zero knowledge even against unbounded queries.

- *Step 2.* We observe that in the IPCPP model the prover can send an oracle proof string π that represents the evaluation table of R, and the verifier can test that π is close to low-degree, and then use self correction to query it. This extension preserves the zero knowledge properties of the previous step.

The more interesting of the two steps is the first one, so we briefly discuss the intuition behind it. Our idea is that, rather than executing the sumcheck protocol on F directly, the prover tells the verifier that $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = z$, then they engage in the sumcheck protocol on the claim $\sum_{\vec{\alpha} \in H^m} \rho F(\vec{\alpha}) + R(\vec{\alpha}) = z$, where ρ is chosen at random by the verifier (after R is sampled). Completeness is clear because if $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$ and $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = z$ then $\sum_{\vec{\alpha} \in H^m} (\rho F + R)(\vec{\alpha}) = z$; soundness is also clear because if $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) \neq 0$ then $\sum_{\vec{\alpha} \in H^m} (\rho F + R)(\vec{\alpha}) \neq z$ with high probability over ρ (regardless of whether $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = z$ or not). We are thus left to show perfect zero knowledge, which turns out to be a much less straightforward argument.

On the surface, perfect zero knowledge appears easy to argue: simply note that $\rho F + R$ is random among all polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$. However, this argument, while compelling, is not enough. First, $\rho F + R$ is *not* random because a malicious verifier can choose ρ depending on queries to R; we discuss this issue further down below. Second, even if $\rho F + R$ were random (e.g., the verifier does not query R before choosing ρ), the simulator must run in polynomial time but it is not clear how that is possible, as we now explain.

Consider the following simulator: (1) sample a random polynomial $Q_{\mathrm{sim}} \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ and use it to simulate $\rho F + R$; (2) whenever the verifier queries $F(\vec{\alpha})$, respond by querying $F(\vec{\alpha})$ and returning the true value; (3) whenever the verifier queries $R(\vec{\alpha})$, respond by querying $F(\vec{\alpha})$ and returning $Q_{\mathrm{sim}}(\vec{\alpha}) - \rho F(\vec{\alpha})$. One can argue that the simulator produces the correct distribution; moreover, the number of queries to F made by the simulator equals the number of (mutually) distinct queries to F and R made by the verifier, as desired.

But how does the simulator sample a random polynomial in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ in polynomial time? The size of the representation of such a polynomial is $\Omega(d^m)$, which is exponential. We get around this problem by exploiting the fact that the number of queries the verifier can make is polynomially bounded, and the simulator can keep state about the answers to past queries and 'make up' on the fly the answer to a new query by resolving dependencies between queries.

More precisely, we leverage our construction of a succinct constraint detector for evaluations of low-degree polynomials (see Section 4.2), which itself relies on tools borrowed from algebraic complexity theory. The same detector also allows to simulate *partial sums*, which the prover sends in the course of the sumcheck protocol itself.

Finally, we explain how we address the issue that the verifier may choose to query R before sending ρ. We handle this by first (implicitly) sampling a random polynomial R_{sim}, and responding to each verifier query to $R(\vec{\alpha})$ with $R_{\mathrm{sim}}(\vec{\alpha})$. Then, when the verifier sends ρ, we draw Q_{sim} conditioned on the already-queried values for R being 'correct'; i.e., for each point $\vec{\alpha}$ queried before ρ is sent, we add the condition that $Q_{\mathrm{sim}}(\vec{\alpha}) = \rho F(\vec{\alpha}) + R_{\mathrm{sim}}(\vec{\alpha})$. We then continue as described above, and it is not too difficult to argue that this strategy yields the correct distribution.

We are now ready to turn the above discussions into formal definitions and proofs. First, we give the definition of the sumcheck relation and of a PZK IPCPP system for sumcheck; then we state and prove the PZK Sumcheck Theorem.

Definition 5.1. *The sumcheck relation and its promise variant are defined as follows.*

- *The* sumcheck relation *is the relation $\mathscr{R}_{\mathrm{SC}}$ of instance-witness pairs $\big((\mathbb{F}, m, d, H, v), F\big)$ such that (i) $\mathbb{F}$ is a finite field, H is a subset of $\mathbb{F}$, v is an element of $\mathbb{F}$, and m, d are positive integers with $\frac{md}{|\mathbb{F}|} < \frac{1}{2}$; (ii) F is in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ and sums to v on H^m.*
- *The* sumcheck promise relation *is the pair of relations $(\mathscr{R}^{\mathsf{YES}}_{\mathrm{SC}}, \mathscr{R}^{\mathsf{NO}}_{\mathrm{SC}})$ where $\mathscr{R}^{\mathsf{YES}}_{\mathrm{SC}} := \mathscr{R}_{\mathrm{SC}}$ and $\mathscr{R}^{\mathsf{NO}}_{\mathrm{SC}}$ are the pairs $\big((\mathbb{F}, m, d, H, v), F\big)$ such that $(\mathbb{F}, m, d, H, v)$ is as above and $F \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ but does <u>not</u> sum to v on H^m.*[8]

Definition 5.2. *A* PZK exact IPCPP system for sumcheck *with soundness error ε is a pair of interactive algorithms (P, V) that satisfies the following properties.*[9]

- COMPLETENESS. *For every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}^{\mathsf{YES}}_{\mathrm{SC}}$, $\Pr\Big[\langle P^F(\mathbb{F}, m, d, H, v), V^F(\mathbb{F}, m, d, H, v)\rangle = 1\Big] = 1$.*
- SOUNDNESS. *For every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}^{\mathsf{NO}}_{\mathrm{SC}}$ and malicious prover $\tilde{P}$, $\Pr\Big[\langle \tilde{P}, V^F(\mathbb{F}, m, d, H, v)\rangle = 1\Big] \leq \varepsilon$.*
- PERFECT ZERO KNOWLEDGE. *There exists a straightline simulator S such that, for every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}^{\mathsf{YES}}_{\mathrm{SC}}$ and malicious verifier $\tilde{V}$, the following two random variables are identically distributed*

$$\Big(S^{\tilde{V}, F}(\mathbb{F}, m, d, H, v)\ ,\ q_S\Big) \quad \text{and} \quad \Big(\mathrm{View}\ \langle P^F(\mathbb{F}, m, d, H, v), \tilde{V}^F\rangle\ ,\ q_{\tilde{V}}\Big)\ ,$$

  where q_S is the number of queries to F made by S and $q_{\tilde{V}}$ is the number of queries to F or the PCP oracle made by $\tilde{V}$. Moreover, S runs in time $\mathrm{poly}(\log|\mathbb{F}| + |H| + m + \mathsf{q}_{\tilde{V}})$, where $\mathsf{q}_{\tilde{V}}$ is $\tilde{V}$'s query complexity.

Theorem 5.3 (PZK Sumcheck)**.** *There exists a PZK public-coin exact IPCPP system (P, V) for the sumcheck promise relation $(\mathscr{R}^{\mathsf{YES}}_{\mathrm{SC}}, \mathscr{R}^{\mathsf{NO}}_{\mathrm{SC}})$ with soundness error $\varepsilon = O(\frac{md}{|\mathbb{F}|})$ and the following efficiency parameters.*

- Oracle round: *P sends an oracle proof string $\pi\colon \mathbb{F}^m \to \mathbb{F}$.*
- Interactive proof: *after the oracle round, P and V engage in an $(m+1)$-round interactive proof; in total, the verifier sends to the prover $O(m)$ field elements, while the prover sends to the verifier $O(md)$ field elements.*
- Queries: *after the interactive proof, V non-adaptively queries π at $\mathrm{poly}(\log|\mathbb{F}| + m + d)$ locations.*
- Space and time: *P runs in space $\mathrm{poly}(\log|\mathbb{F}| + m + d + |H|)$, while V in time $\mathrm{poly}(\log|\mathbb{F}| + m + d + |H|)$. (The prover's space complexity assumes that the randomness tape is two-way rather than one-way; see Remark 5.7 below.)*

5.1 Step 1

We construct a public-coin IP for sumcheck that is perfect zero knowledge, in the "R-hybrid" model, where the prover and verifier have access to a uniformly random $R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$.

Construction 5.4. *The IP system $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$ is defined as follows. Both P_{IP} and V_{IP} receive a tuple $(\mathbb{F}, m, d, H, v)$ as common input, and two polynomials $F, R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ as oracles. The interaction proceeds as follows:*

[8] This promise is *not* the same notion as in Section 3.2.3; there the promise is with respect to instances, whereas here it is with respect to witnesses.

[9] This is exactly the standard definition of an IPCPP (Section 3.3.2), but with a soundness condition respecting our current notion of a promise.

1. *P_{IP} sends $z := \sum_{\vec{\alpha} \in H^m} R(\vec{\alpha})$ to V_{IP};*
2. *V_{IP} draws a random element ρ in $\mathbb{F}$, and sends ρ to P_{IP};*
3. *P_{IP} and V_{IP} run the sumcheck IP [LFKN92, Sha92] on the statement "$\sum_{\vec{\alpha} \in H^m} Q(\vec{\alpha}) = \rho v + z$" where $Q := \rho F + R$ (with P_{IP} playing the role of the prover and V_{IP} that of the verifier).*

Note that $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$ is public-coin, and satisfies the following efficiency properties.

– Communication: The number of rounds is $m + 1$. Across the interaction, V_{IP} sends $O(m)$ field elements to P_{IP}, while P_{IP} sends $O(md)$ field elements to V_{IP}.

– Queries: V_{IP} queries F and R each at a single random point because, at the end of the sumcheck protocol, the verifier queries Q at a random point $\vec{\gamma}$, and such a query can be "simulated" by querying F and R at $\vec{\gamma}$ and then using these answers, along with ρ, to compute the necessary value for Q.

– Space and time: P_{IP} runs in space $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H|)$, while V_{IP} in time $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H|)$. (The prover's space complexity assumes that the randomness tape is two-way; see Remark 5.7 below.)

We now state and prove the completeness, soundness, and perfect zero knowledge properties.

Lemma 5.5. *The IP system $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$ satisfies the following properties.*

- COMPLETENESS. *For every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{YES}}$ and $R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ with $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = 0$,*

$$\Pr\Big[\langle P_{\mathrm{IP}}^{F,R}(\mathbb{F}, m, d, H, v), V_{\mathrm{IP}}^{F,R}(\mathbb{F}, m, d, H, v)\rangle = 1\Big] = 1 \ .$$

- SOUNDNESS. *For every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{NO}}$, $R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$, and malicious prover $\tilde{P}$,*

$$\Pr\Big[\langle \tilde{P}, V_{\mathrm{IP}}^{F,R}(\mathbb{F}, m, d, H, v)\rangle = 1\Big] \leq \frac{md+1}{|\mathbb{F}|} \ .$$

- PERFECT ZERO KNOWLEDGE. *There exists a straightline simulator S_{IP} such that, for every $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{YES}}$ and malicious verifier $\tilde{V}$, the following two random variables are identically distributed*

$$\Big(S_{\mathrm{IP}}^{\tilde{V},F}(\mathbb{F}, m, d, H, v) \ , \ q_{S_{\mathrm{IP}}}\Big) \quad \textit{and} \quad \Big(\mathrm{View}\ \langle P_{\mathrm{IP}}^{F,R}(\mathbb{F}, m, d, H, v), \tilde{V}^{F,R}\rangle \ , \ q_{\tilde{V}}\Big) \ ,$$

where R is uniformly random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$, $q_{S_{\mathrm{IP}}}$ is the number of queries to F made by S_{IP}, and $q_{\tilde{V}}$ is the number of queries to F or R made by $\tilde{V}$. Moreover, S_{IP} runs in time $\mathrm{poly}(\log |\mathbb{F}| + m + d + |H| + \mathsf{q}_{\tilde{V}})$ where $\mathsf{q}_{\tilde{V}}$ is $\tilde{V}$'s query complexity.

Proof. We argue first completeness, then soundness, and, finally, perfect zero knowledge.

Completeness. If both F sums to v on H^m and R sums to z on H^m, then $Q := \rho F + R$ sums to $\rho v + z$ on H^m for every choice of ρ. Then completeness follows from the completeness of standard sumcheck.

Soundness. For every $F, R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ with $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) \neq v$, $\sum_{\vec{\alpha} \in H^m} Q(\vec{\alpha})$ equals $\rho v + z$ for at most one choice of ρ, namely, $(\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) - z)/(v - \sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}))$. Thus, except with probability $1/|\mathbb{F}|$, the sumcheck protocol is invoked on an incorrect claim, which incurs a soundness error of at most $\frac{md}{|\mathbb{F}|}$. The claimed soundness error follows by a union bound.

Perfect zero knowledge. We begin by proving perfect zero knowledge via a straightline simulator S_{slow} whose number of queries to F equals $q_{\tilde{V}}$, but runs in time $\mathrm{poly}(|\mathbb{F}|^m + q_{\tilde{V}})$. After that, we explain how to modify S_{slow} into another simulator S_{IP}, with an identical output distribution, that runs in the faster time claimed in the lemma.

The simulator S_{slow}, given straightline access to $\tilde{V}$ and oracle access to F, works as follows:

1. Draw a uniformly random $R_{\text{sim}} \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ and send $z_{\text{sim}} := \sum_{\vec{\alpha} \in H^m} R_{\text{sim}}(\vec{\alpha})$ to $\tilde{V}$.
2. Whenever $\tilde{V}$ queries F at $\vec{\gamma} \in \mathbb{F}^m$, return $F(\vec{\gamma})$; whenever $\tilde{V}$ queries R at $\vec{\gamma} \in \mathbb{F}^m$, return $R_{\text{sim}}(\vec{\gamma})$.
3. Receive $\tilde{\rho}$ from $\tilde{V}$, and draw a uniformly random $Q_{\text{sim}} \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ conditioned on $\sum_{\vec{\alpha} \in H^m} Q_{\text{sim}}(\vec{\alpha}) = \tilde{\rho} v + z_{\text{sim}}$ and $Q_{\text{sim}}(\vec{\gamma}) = \tilde{\rho} F(\vec{\gamma}) + R_{\text{sim}}(\vec{\gamma})$ for every coordinate $\vec{\gamma} \in \mathbb{F}^m$ queried in Step 2. (This latter condition requires querying F at $\vec{\gamma}$ for every coordinate $\vec{\gamma} \in \mathbb{F}^m$ queried to R_{sim} in Step 2.)
4. Hereafter: whenever $\tilde{V}$ queries F at $\vec{\gamma} \in \mathbb{F}^m$, return $F(\vec{\gamma})$; whenever $\tilde{V}$ queries R at $\vec{\gamma} \in \mathbb{F}^m$, return $Q_{\text{sim}}(\vec{\gamma}) - \tilde{\rho} F(\vec{\gamma})$. (In either case, a query to F is required.)
5. Run the sumcheck protocol with $\tilde{V}$ on Q_{sim}. (Note that $\tilde{V}$ may query F or R before, during, or after this protocol.)
6. Output the view of the simulated $\tilde{V}$.

Note that S_{slow} runs in time $\text{poly}(|\mathbb{F}|^m + q_{\tilde{V}})$. Also, S_{slow} makes one query to F for every query to F or R by $\tilde{V}$ (at least provided that $\tilde{V}$'s queries have no duplicates, which we can assume without loss of generality). Thus, overall, the number of queries to F by S_{slow} is $q_{\tilde{V}}$. We now argue that S_{slow}'s output is identically distributed to $\tilde{V}$'s view when interacting with the honest prover P_{IP}, for R random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$.

Claim. $S_{\text{slow}}^{\tilde{V},F} \equiv \text{View}\ \langle P_{\text{IP}}^{F,R}, \tilde{V}^{F,R} \rangle$.

Proof. Define the random variable $Q := \tilde{\rho} F + R$, where $\tilde{\rho}$ is chosen by $\tilde{V}$. Observe that there exists a (deterministic) function $v(\cdot)$ such that

$$\text{View}\ \langle P_{\text{IP}}^{F,R}, \tilde{V}^{F,R} \rangle = v(Q, F, r) \quad \text{and} \quad S_{\text{slow}}^{\tilde{V},F} = v(Q_{\text{sim}}, F, r)\ ,$$

where the random variable r is $\tilde{V}$'s private randomness. Indeed, (i) the messages sent and received by $\tilde{V}$ are identical to those when interacting with P_{IP} on Q and Q_{sim}, respectively; (ii) $\tilde{V}$'s queries to F are answered honestly; (iii) $\tilde{V}$'s queries to R are answered by $R = Q - \tilde{\rho} F$ and $R_{\text{sim}} = Q_{\text{sim}} - \tilde{\rho} F$ respectively. We are only left to argue that, for any choice of r, Q and Q_{sim} are identically distributed:

- $Q = \tilde{\rho} F + R$ is uniformly random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ conditioned on $\sum_{\vec{\alpha} \in H^m} Q(\vec{\alpha}) = \tilde{\rho} v + z$, because R is uniformly random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ and satisfies $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = z$ (and F is in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ and satisfies $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = v$); and
- Q_{sim} is uniformly random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ conditioned on $\sum_{\vec{\alpha} \in H^m} Q_{\text{sim}}(\vec{\alpha}) = \tilde{\rho} v + z_{\text{sim}}$, because Q_{sim} is sampled at random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ conditioned on $\sum_{\vec{\alpha} \in H^m} Q_{\text{sim}}(\vec{\alpha}) = \tilde{\rho} v + z_{\text{sim}}$ and $Q_{\text{sim}}(\vec{\gamma}_i) = R_{\text{sim}}(\vec{\gamma}_i) + \tilde{\rho} F(\vec{\gamma}_i)$ for some (adversarial) choice of $\vec{\gamma}_1, \ldots, \vec{\gamma}_k$. But R_{sim} is uniformly random in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$, so the latter condition says that Q_{sim} matches a random polynomial on the set of points $\{\vec{\gamma}_1, \ldots, \vec{\gamma}_k\}$, giving the claimed distribution for Q_{sim}. □

We explain how to modify S_{slow} so as to reduce the running time to $\text{poly}(\log |\mathbb{F}| + m + d + |H| + \mathtt{q}_{\tilde{V}})$.

Note that S_{slow}'s inefficiency arises from sampling two random polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$, namely R_{sim} and Q_{sim}, subject to certain constraints, and using them to answer $\tilde{V}$'s messages and queries. We observe (and carefully justify below) that all information about R_{sim} and Q_{sim} received by $\tilde{V}$ is answers to queries of the form "given $\vec{\gamma} \in \mathbb{F}^{\leq m}$, return the value $A(\vec{\gamma}) := \sum_{\vec{\alpha} \in H^{m-|\vec{\gamma}|}} A(\vec{\gamma}, \vec{\alpha})$" for a random $A \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$, possibly conditioned on previous such queries; when $\vec{\gamma}$ has length zero we use the symbol $\perp$, so that $A(\perp)$ denotes $\sum_{\vec{\alpha} \in H^m} A(\vec{\alpha})$. The new simulator can use the algorithm $\mathcal{A}$ from our Corollary 4.10 to adaptively answer such queries, without ever explicitly sampling the two polynomials.

We now argue that all information about R_{sim} and Q_{sim} received by $\tilde{V}$ from S_{slow} can be viewed as queries of the above form, by discussing each step of S_{slow}.

- In Step 1, S_{slow} draws a uniformly random $R_{\text{sim}} \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ and sends $z_{\text{sim}} = R_{\text{sim}}(\perp)$.
- In Step 2, S_{slow} answers any query $\vec{\gamma} \in \mathbb{F}^m$ to R with $R_{\text{sim}}(\vec{\gamma})$.

- In Step 3, S_{slow} draws a uniformly random $Q_{\text{sim}} \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ conditioned on $Q_{\text{sim}}(\perp) = \tilde{\rho} v + z_{\text{sim}}$ and also on $Q_{\text{sim}}(\vec{\gamma}) = R_{\text{sim}}(\vec{\gamma}) + \tilde{\rho} F(\vec{\gamma})$ for at most $q_{\tilde{V}}$ points $\vec{\gamma} \in \mathbb{F}^m$ (namely, the points corresponding to queries in Step 2).

- In Step 4, S_{slow} replies any query $\vec{\gamma} \in \mathbb{F}^m$ to R with $Q_{\text{sim}}(\vec{\gamma}) - \tilde{\rho} F(\vec{\gamma})$.

- In Step 5, S_{slow} runs the sumcheck protocol with $\tilde{V}$ on Q_{sim}, which requires computing univariate polynomials of the form $\sum_{\vec{\alpha} \in H^{m-|\theta|-1}} Q_{\text{sim}}(\vec{\theta}, X, \vec{\alpha}) \in \mathbb{F}[X]$ for various choices of $\vec{\theta} \in \mathbb{F}^{<m}$. Each of these polynomials has degree less than d, and so can be obtained by interpolation from its evaluation at any d distinct points; each of these is the answer of a query $Q_{\text{sim}}(\vec{\gamma})$ of the required form, with $\vec{\gamma} = (\vec{\theta}, \delta)$ for some $\delta \in \mathbb{F}$. Overall, during the protocol, S_{slow} only needs to query Q_{sim} at md points $\vec{\gamma} \in \mathbb{F}^{\leq m}$.

In sum, we can modify S_{slow} so that instead of explicitly sampling R_{sim} and Q_{sim}, it uses $\mathcal{A}$ to sample the answer for each query to Q_{sim} or R_{sim}, conditioning the uniform distribution on the answers to previous queries. Putting all of this together, we obtain the simulator S_{IP} described below, whose output is identically distributed to the output of S_{slow}.

> The simulator S_{IP}, given straightline access to $\tilde{V}$ and oracle access to F, works as follows:
>
> 1. Let $\mathsf{ans}_{R_{\text{sim}}}$ be a subset of $\mathbb{F}^{\leq m} \times \mathbb{F}$ that records query-value pairs for R_{sim}.
> 2. Whenever $\tilde{V}$ queries F at $\vec{\gamma} \in \mathbb{F}^m$, return $F(\vec{\gamma})$; whenever $\tilde{V}$ queries R at $\vec{\gamma} \in \mathbb{F}^m$, return $\beta := \mathcal{A}(\mathbb{F}, m, d, H, \mathsf{ans}_{R_{\text{sim}}}, \vec{\gamma})$. In the latter case, add $(\vec{\gamma}, \beta)$ to $\mathsf{ans}_{R_{\text{sim}}}$.
> 3. Send $z_{\text{sim}} := \mathcal{A}(\mathbb{F}, m, d, H, \mathsf{ans}_{R_{\text{sim}}}, \perp)$, and add $(\perp, z_{\text{sim}})$ to $\mathsf{ans}_{R_{\text{sim}}}$.
> 4. Receive $\tilde{\rho}$ from $\tilde{V}$, and compute $\mathsf{ans}_{Q_{\text{sim}}} := \{(\vec{\gamma}, \beta + \tilde{\rho} F(\vec{\gamma}))\}_{(\vec{\gamma}, \beta) \in \mathsf{ans}_{R_{\text{sim}}}}$; this subset of $\mathbb{F}^{\leq m} \times \mathbb{F}$ records query-value pairs for Q_{sim}. Note that $\mathsf{ans}_{Q_{\text{sim}}}$ includes the pair $(\perp, \tilde{\rho} v + z_{\text{sim}})$ because $F(\perp) = v$ by assumption.
> 5. Hereafter: whenever $\tilde{V}$ queries F at $\vec{\gamma} \in \mathbb{F}^m$, return $F(\vec{\gamma})$; whenever $\tilde{V}$ queries R at $\vec{\gamma} \in \mathbb{F}^m$, return $\beta' := \beta - \tilde{\rho} F(\vec{\gamma})$ where $\beta := \mathcal{A}(\mathbb{F}, m, d, H, \mathsf{ans}_{Q_{\text{sim}}}, \vec{\gamma})$. In the latter case, add $(\vec{\gamma}, \beta)$ to $\mathsf{ans}_{Q_{\text{sim}}}$.
> 6. Run the sumcheck protocol with $\tilde{V}$ on Q_{sim}, by using the algorithm $\mathcal{A}$ and updating $\mathsf{ans}_{Q_{\text{sim}}}$ appropriately. (Note that $\tilde{V}$ may query F or R before, during, or after this protocol.)
> 7. Output the view of the simulated $\tilde{V}$.

Note that S_{IP} makes the same number of queries to F as S_{slow} does. Also, the number of pairs in $\mathsf{ans}_{R_{\text{sim}}}$ is at most $q_{\tilde{V}} + md + 1$; ditto for $\mathsf{ans}_{Q_{\text{sim}}}$. Since the algorithm $\mathcal{A}$ is called at most $q_{\tilde{V}} + md$ times, the running time of S_{IP} is $\text{poly}(\log |\mathbb{F}| + m + d + |H| + \mathsf{q}_{\tilde{V}})$, as required. □

5.2 Step 2

The IP described and analyzed in Section 5.1 is in the "R-hybrid" model. We now compile that IP into an IPCPP, by using proximity testing and self-correction, thereby concluding the proof of the PZK Sumcheck Theorem.

Proof of Theorem 5.3. Construct an IPCPP system (P, V) for sumcheck as follows:

- The prover P, given input $(\mathbb{F}, m, d, H, v)$ and oracle access to F, samples a uniformly random polynomial $R \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ and sends its evaluation $\pi \colon \mathbb{F}^m \to \mathbb{F}$ to the verifier V. Then P simulates $P_{\text{IP}}^{F,R}(\mathbb{F}, m, d, H, v)$.

- The verifier V, after receiving a proof string $\pi \colon \mathbb{F}^m \to \mathbb{F}$, simulates $V_{\text{IP}}^{F,\pi}(\mathbb{F}, m, d, H, v)$ up to V_{IP}'s single query $\vec{\alpha} \in \mathbb{F}^m$ to π (which occurs after the interaction), which V does not answer directly but instead answers as follows. First, V checks that π is ϱ-close to the evaluation of a polynomial in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ by performing an individual-degree test with proximity parameter $\varrho := \frac{1}{8}$ and soundness error $\epsilon := \frac{md}{|\mathbb{F}|}$ [GS06, GR15]; then, V computes $\pi(\vec{\alpha})$ via self-correction with soundness error ϵ [RS96, AS03], and replies with that value. Both procedures require $\text{poly}(\log |\mathbb{F}| + m + d)$ queries and time. Finally, V rejects if V_{IP} rejects or the individual degree test rejects.

Completeness and perfect zero knowledge of (P, V) are inherited, in a straightforward way, from those of $(P_{\text{IP}}, V_{\text{IP}})$. We now argue soundness. So consider an instance-witness pair $\big((\mathbb{F}, m, d, H, v), F\big) \in \mathscr{R}_{\text{SC}}^{\text{NO}}$ and a malicious prover $\tilde{P}$, and denote by $\tilde{\pi} \colon \mathbb{F}^m \to \mathbb{F}$ the proof string sent by $\tilde{P}$. We distinguish between the following two cases.

- *Case 1: $\tilde{\pi}$ is ϱ-far from evaluations of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$.*

  In this case, the low-degree test accepts with probability at most ϵ.

- *Case 2: $\tilde{\pi}$ is ϱ-close to evaluations of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$.*

  In this case, let $\tilde{R}$ be the unique polynomial in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$ whose evaluation is ϱ-close to $\tilde{\pi}$; this polynomial exists because ϱ is less than the unique decoding radius (of the corresponding Reed–Muller code), which equals $\frac{1}{2}(1 - \frac{d-1}{|\mathbb{F}|})^m$, and is at least $\frac{1}{4}$ by the assumption that $\frac{md}{|\mathbb{F}|} < \frac{1}{2}$. By the soundness of $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$, the probability that $V_{\mathrm{IP}}^{F,\tilde{R}}$ accepts is at most $\frac{md+1}{|\mathbb{F}|}$ (see Lemma 5.5). However V only has access to $\tilde{\pi}$, and uses self-correction on it to compute $\tilde{R}$ at the single location $\vec{\alpha} \in \mathbb{F}^m$ required by V_{IP}; the probability that the returned value is not correct is at most ϵ. Hence, by a union bound, V accepts with probability at most $\frac{md+1}{|\mathbb{F}|} + \epsilon$.

Overall, we deduce that V accepts with probability at most $\max\{\epsilon\,,\ \frac{md+1}{|\mathbb{F}|} + \epsilon\} \leq 3\frac{md}{|\mathbb{F}|}$. □

Remark 5.6 (is interaction needed?)**.** One may be tempted to "flatten" the IPCPP used to prove Theorem 5.3, by sending a single PCP that already contains all possible transcripts, relative to all possible ρ's. Such a modification does indeed preserve completeness and soundness. (In fact, even a small subset of ρ's is enough for constant soundness error, because only one ρ in $\mathbb{F}$ is "bad".) However, this modification does *not* preserve zero knowledge: if a verifier learns, say, the partial sums $\alpha_1 := \rho_1 F(\vec{\gamma}) + R(\vec{\gamma})$ and $\alpha_2 := \rho_2 F(\vec{\gamma}) + R(\vec{\gamma})$ for $\rho_1 \neq \rho_2$ and some $\vec{\gamma} \in \mathbb{F}^{\leq m}$ then he also learns $F(\vec{\gamma}) = \frac{\alpha_1 - \alpha_2}{\rho_1 - \rho_2}$, violating zero knowledge. (Yet, the modification *does* preserve *honest-verifier* zero knowledge.)

Remark 5.7 (space complexity of the prover)**.** The prover in a zero knowledge protocol is a probabilistic function, and hence reads bits from its randomness tape. In the case of the above protocol, the prover P must sample the evaluation of a random polynomial R in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$; the entropy of R is exponential and thus requires reading an exponential number of random bits from the randomness tape. (Beyond this, P requires no other randomness.)

It is easy to see that P can run in exponential time and space. However, if the prover has *two*-way access to its random tape, P can run in exponential time and polynomial space: the prover treats the random tape as the coefficients of R, computing an evaluation at a given point by reading the tape coefficient-by-coefficient and summing the contributions of each monomial.

Two-way access to the randomness tape is a relaxation of the standard definition, which permits only *one*-way access to it [Nis93]; that is, random bits must be stored on the work tape in order to be accessed again. It is not known whether the relaxation makes polynomial-space machines more powerful for decision problems (the class is equivalent to "almost"-$\mathbf{PSPACE}$ [BVW98]), nor do we know how to obtain a polynomial-space prover with only one-way access. Nevertheless, we believe that polynomial space with two-way access to the random tape is still quite meaningful, e.g., it yields standard polynomial space relative to a random oracle.

6 Perfect zero knowledge for counting problems

We prove that $\#\mathbf{P}$ has an IPCP that is perfect zero knowledge against unbounded queries. (Recall that $\#\mathbf{P}$ corresponds to all counting problems associated to decision problems in $\mathbf{NP}$.) We do so by constructing a suitable protocol for the counting problem associated to 3SAT, which is $\#\mathbf{P}$-complete.

Definition 6.1. *Let $\mathscr{L}_{\#3\mathrm{SAT}}$ be the language of pairs (ϕ, N) where ϕ is a 3-CNF boolean formula and N is the number of satisfying assignments of ϕ. We denote by n the number of variables and by c the number of clauses in ϕ.*

We construct a public-coin IPCP system for $\mathscr{L}_{\#3\mathrm{SAT}}$ that is perfect zero knowledge against unbounded queries, and has exponential proof length and polynomial query complexity. As in the non-ZK IP counterpart, the number of rounds is $O(n)$, the prover runs in space $\mathrm{poly}(c)$ (with a caveat, see Remark 5.7), and the verifier in time $\mathrm{poly}(c)$.

Theorem 6.2 (formal statement of 1.2)**.** *There exists a IPCP system (P, V) that puts $\mathscr{L}_{\#3\mathrm{SAT}}$ in the complexity class*

$$\mathbf{PZK\text{-}IPCP}\left[\begin{array}{llll} \text{rounds} & \mathsf{k} & = & O(n) \\ \text{proof length} & \mathsf{l} & = & \exp(n) \\ \text{query complexity} & \mathsf{q} & = & \mathrm{poly}(c) \\ \text{soundness error} & \varepsilon & = & 1/2 \\ \text{prover space} & \mathsf{sp} & = & \mathrm{poly}(c) \\ \text{verifier time} & \mathsf{tv} & = & \mathrm{poly}(c) \\ \text{query bound} & \mathsf{b} & = & * \end{array}\right].$$

Moreover, the verifier V is public-coin and non-adaptive.

Proof. Let $(P_{\mathrm{SC}}, V_{\mathrm{SC}})$ be the PZK IPCPP system for sumcheck from Theorem 5.3, and let S_{SC} be any simulator attesting to its perfect zero knowledge. We construct an IPCP system (P, V) for $\mathscr{L}_{\#3\mathrm{SAT}}$ as follows.

- The prover P, given an instance (ϕ, N), finds a prime $q \in (2^n, 2^{2n}]$, computes the arithmetization $p_\phi \in \mathbb{F}_q^{<3c}[X_1, \ldots, X_n]$ of ϕ and simulates $P_{\mathrm{SC}}^F(\mathbb{F}, m, d, H, v)$ with $\mathbb{F} := \mathbb{F}_q$, $m := n$, $d := 3c$, $H := \{0, 1\}$, $v := N$, and $F(X_1, \ldots, X_n) := p_\phi(X_1, \ldots, X_n)$. (The prover P also communicates the prime q to the V verifier.)
- The verifier V, given an instance (ϕ, N), also computes $\mathbb{F}, m, d, H, v, F$, and then simulates $V_{\mathrm{SC}}^F(\mathbb{F}, m, d, H, v)$, and accepts if and only if the simulation accepts.

Note that the arithmetization of a 3-CNF formula ϕ can be computed in time $\mathrm{poly}(c)$, and the claimed given efficiency parameters follow from Theorem 5.3. We now argue completeness, then soundness, and finally perfect zero knowledge.

Completeness. Completeness follows from the completeness of $(P_{\mathrm{SC}}, V_{\mathrm{SC}})$ and the fact that if $(\phi, N) \in \mathscr{L}_{\#3\mathrm{SAT}}$ then $\big((\mathbb{F}, m, d, H, v), F\big) = ((\mathbb{F}_q, n, 3c, \{0,1\}^n, N), p_\phi) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{YES}}$.

Soundness. Soundness follows from the soundness of $(P_{\mathrm{SC}}, V_{\mathrm{SC}})$ and the fact that if $(\phi, N) \notin \mathscr{L}_{\#3\mathrm{SAT}}$ then $\big((\mathbb{F}, m, d, H, v), F\big) = ((\mathbb{F}_q, n, 3c, \{0,1\}^n, N), p_\phi) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{NO}}$.

Perfect zero knowledge. We construct a simulator S that provides perfect zero knowledge. Given an instance (ϕ, N) and straightline access to a verifier $\tilde{V}$, the simulator S computes $\mathbb{F}, m, d, H, v, F$ as above and simulates $S_{\mathrm{SC}}^{\tilde{V}, F}(\mathbb{F}, m, d, H, v)$. By the perfect zero knowledge property of $(P_{\mathrm{SC}}, V_{\mathrm{SC}})$, the simulator's output is identically distributed to $\mathrm{View}\,\langle P_{\mathrm{SC}}^F(\mathbb{F}, m, d, H, v), \tilde{V}^F\rangle$; but note that $\tilde{V}$ does not query any oracles outside of its interaction with P_{SC} so that $\tilde{V}^F = \tilde{V}$. By the construction of P above, this view equals $\mathrm{View}\,\langle P(\phi, N), \tilde{V}\rangle$, as desired. □

7 Perfect zero knowledge from succinct constraint detection

We show how to obtain perfect zero knowledge 2-round IOPs of Proximity for *any* linear code that has proximity proofs with succinct constraint detection (Section 7.1). Afterwards, we instantiate this general transformation for the case of Reed–Solomon codes (Section 7.2), whose proximity proofs we discussed in Section 4.3.

7.1 A general transformation

PCPs of proximity for codes. A *PCP of Proximity* [DR04, BGH+06] for a code family $\mathscr{C} = \{C_{\mathtt{n}}\}_{\mathtt{n}}$ is a pair $(P_{\mathscr{C}}, V_{\mathscr{C}})$ where for every index $\mathtt{n}$ and $w \in \mathbb{F}^{D(\mathtt{n})}$: if $w \in C_{\mathtt{n}}$ then $V_{\mathscr{C}}^{w,\pi}(\mathtt{n})$ accepts with probability 1 with $\pi := P(\mathtt{n}, w)$; if w is 'far' from $C_{\mathtt{n}}$ then $w \in C_{\mathtt{n}}$ rejects with high probability regardless of π. We do not formally define this notion because it is a special case of a 1-round IOP of Proximity (see Section 3.2.3) where the verifier message is empty; we use $\mathbf{PCPP}[\mathsf{l}, \mathsf{q}, \varepsilon, \delta, \mathsf{tp}, \mathsf{tv}]$ to denote the corresponding complexity class. Note that, since both π and w are provided as oracles to V, the query complexity of V is the *total* number of queries across both oracles.

Leakage from proximity proofs. While proximity proofs facilitate local testing, they are a liability for zero knowledge: in principle even a single query to π may 'summarize' information that needs many queries to w to simulate. (This holds for BS proximity proofs [BS08], for instance.) Our construction facilitates local testing while avoiding this leakage.

Perfect zero knowledge IOPs of Proximity. The notion of zero knowledge for IOPs of Proximity that we target is defined in Section 3.3.2 and is analogous to [IW14]'s notion for PCPs of Proximity (a special case of our setting). Informally, it requiress an algorithm that simulates the verifier's view by making as many queries to w as the *total* number of verifier queries to either w or any oracles sent by the prover; intuitively, this means that any bit of any message oracle reveals no more information than one bit of w.

A generic 'masking' construction. Suppose that the linear code family $\mathscr{C} = \{C_{\mathtt{n}}\}_{\mathtt{n}}$ has a PCP of Proximity. Consider the 2-round IOP of Proximity that uses masking via random self-reducibility (similarly to [BCGV16]) as follows. The prover and verifier have input $\mathtt{n}$ and oracle access to a codeword w, and the prover wants to convince the verifier that w is close to $C_{\mathtt{n}}$. Rather than sending a proximity proof for w, the prover samples a random codeword $z \in C_{\mathtt{n}}$ and sends it to the verifier; the verifier replies with a random field element ρ; the prover sends a proximity proof for the new codeword $\rho w + z$. Completeness follows from linearity of $C_{\mathtt{n}}$; soundness follows from the fact that if w is far from $C_{\mathtt{n}}$ then so is the word $\rho w + z$ for every z with high probability over ρ.

Perfect zero knowledge intuitively follows from the observation that $\rho w + z$ is essentially a random codeword (up to malicious choice of ρ). We formally prove this for the case where *the linear code family consisting of the concatenation of codewords in $\mathscr{C}$ with corresponding proximity proofs has succinct constraint detection*.

We are now ready to turn the above discussions into formal definitions and proofs. Throughout, given a code family $\mathscr{C}$, we denote by $\mathrm{Rel}(\mathscr{C})$ the relation consisting of all pairs $(\mathtt{n}, w)$ such that $w \in C_{\mathtt{n}}$.

Definition 7.1. *Let $\mathscr{C} = \{C_{\mathtt{n}}\}_{\mathtt{n}}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$, and let (P, V) be a PCPP system for $\mathrm{Rel}(\mathscr{C})$. We say that (P, V) is* **linear** *if P is deterministic and is linear in its input codeword: for every index $\mathtt{n}$ there exists a matrix $A_{\mathtt{n}}$ with entries in $\mathbb{F}(\mathtt{n})$ such that $P(\mathtt{n}, w) = A_{\mathtt{n}} \cdot w$ for all $w \in C_{\mathtt{n}}$ (equivalently, the set $\{\, w \| P(\mathtt{n}, w) \,\}_{w \in C_{\mathtt{n}}}$ is a linear code).*

Definition 7.2. *Let $\mathscr{C} = \{C_{\mathtt{n}}\}_{\mathtt{n}}$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$, and let (P, V) be a PCPP system for $\mathrm{Rel}(\mathscr{C})$. We say that (P, V) has $T(\cdot, \cdot)$-***time constraint detection*** *if (P, V) is linear and, moreover, the linear code family $\mathscr{L} = \{L_{\mathtt{n}}\}$ has $T(\cdot, \cdot)$-time constraint detection, where $L_{\mathtt{n}} := \{\, w \| P(\mathtt{n}, w) \,\}_{w \in C_{\mathtt{n}}}$; we also say that (P, V) has* **succinct constraint detection** *if the same holds with $T(\mathtt{n}, \ell) = \mathrm{poly}(\mathtt{n} + \ell)$.*

Theorem 7.3. *Let $\mathscr{C} = \{C_{\mathtt{n}}\}_{\mathtt{n}}$ be a linear code family with domain $D(\cdot)$, alphabet $\mathbb{F}(\cdot)$, block length $\ell(\cdot)$, and a $S(\cdot)$-time sampler. Suppose that there exists a PCPP system $(P_{\mathscr{C}}, V_{\mathscr{C}})$ for $\mathrm{Rel}(\mathscr{C})$ that (is linear and) has succinct*

constraint detection and puts $\mathrm{Rel}(\mathscr{C})$ *in the complexity class*

$$\mathbf{PCPP}\left[\begin{array}{ll} \text{answer alphabet} & \mathbb{F}(\mathfrak{n}) \\ \text{proof length} & \mathsf{l}_{\mathscr{C}}(\mathfrak{n}) \\ \text{query complexity} & \mathsf{q}_{\mathscr{C}}(\mathfrak{n}) \\ \text{soundness error} & \varepsilon_{\mathscr{C}}(\mathfrak{n}) \\ \text{proximity parameter} & \delta_{\mathscr{C}}(\mathfrak{n}) \\ \text{prover time} & \mathsf{tp}_{\mathscr{C}}(\mathfrak{n}) \\ \text{verifier time} & \mathsf{tv}_{\mathscr{C}}(\mathfrak{n}) \\ \text{verifier randomness} & \mathsf{rv}_{\mathscr{C}}(\mathfrak{n}) \end{array}\right].$$

Then there exists an IOPP system (P, V) *that puts* $\mathrm{Rel}(\mathscr{C})$ *in the complexity class*

$$\mathbf{PZK\text{-}IOPP}\left[\begin{array}{lllll} \text{answer alphabet} & \mathbb{F}(\mathfrak{n}) & & & \\ \text{rounds} & \mathsf{k}(\mathfrak{n}) & = & 2 \\ \text{proof length} & \mathsf{l}(\mathfrak{n}) & = & \mathsf{l}_{\mathscr{C}}(\mathfrak{n}) + \ell(\mathfrak{n}) \\ \text{query complexity} & \mathsf{q}(\mathfrak{n}) & = & 2\mathsf{q}_{\mathscr{C}}(\mathfrak{n}) \\ \text{soundness error} & \varepsilon(\mathfrak{n}) & = & \varepsilon_{\mathscr{C}}(\mathfrak{n}) + \frac{1}{|\mathbb{F}(\mathfrak{n})|} \\ \text{proximity parameter} & \delta(\mathfrak{n}) & = & 2\delta_{\mathscr{C}}(\mathfrak{n}) \\ \text{prover time} & \mathsf{tp}(\mathfrak{n}) & = & \mathsf{tp}_{\mathscr{C}}(\mathfrak{n}) + S(\mathfrak{n}) + O(\ell(\mathfrak{n})) \\ \text{verifier time} & \mathsf{tv}(\mathfrak{n}) & = & \mathsf{tv}_{\mathscr{C}}(\mathfrak{n}) + O(\mathsf{q}_{\mathscr{C}}(\mathfrak{n})) \\ \text{verifier randomness} & \mathsf{rv}(\mathfrak{n}) & = & \mathsf{rv}_{\mathscr{C}}(\mathfrak{n}) + \log|\mathbb{F}(\mathfrak{n})| \\ \text{query bound} & \mathsf{b}(\mathfrak{n}) & = & * \end{array}\right].$$

Remark 7.4 (the case of LTCs)**.** It is tempting to apply Theorem 7.3 to the notable special case where the proximity proof is *empty* (e.g., when $\mathscr{C}$ is locally testable so no proximity proofs are needed). However, in this case, the zero knowledge guarantee of our construction does not buy anything compared to when the verifier queries only the codeword (indeed, the verifier *already* learns precisely the value of codeword at those positions which it queries and nothing else).

Construction 7.5. *The IOPP system* (P, V) *is defined as follows. The prover receives a pair* $(\mathfrak{n}, w)$ *as input, while the verifier receives the index* $\mathfrak{n}$ *as input and* w *as an oracle. The interaction proceeds as follows:*

1. *P samples a random codeword z in $C_{\mathfrak{n}}$ and sends z to V;*
2. *V samples a random element ρ in $\mathbb{F}(\mathfrak{n})$ and sends ρ to P;*
3. *P computes the proof $\pi := P_{\mathscr{C}}(\mathfrak{n}, \rho w + z)$ and sends π to V;*
4. *V checks that $V_{\mathscr{C}}^{w',\pi}(\mathfrak{n})$ accepts, where $w' := \rho w + z$ (any query α to w' is computed as $\rho w(\alpha) + z(\alpha)$).*

The round complexity, proof length, query complexity, and prover and verifier complexities claimed in Theorem 7.3 follow in a straightforward way from Construction 7.5. We now argue completeness and soundness (Claim 7.6) and perfect zero knowledge (Claim 7.7).

Claim 7.6. *The IOPP system* (P, V) *has completeness* 1 *and soundness error* $\varepsilon(\mathfrak{n}) = \varepsilon_{\mathscr{C}}(\mathfrak{n}) + \frac{1}{|\mathbb{F}(\mathfrak{n})|}$.

Proof. First we argue completeness. Suppose that the instance-witness pair $(\mathfrak{n}, w)$ is in the relation $\mathrm{Rel}(\mathscr{C})$, i.e., that the word w is in the code $C_{\mathfrak{n}}$. Then, the linearity of $C_{\mathfrak{n}}$ implies that, for every word z in $C_{\mathfrak{n}}$ and element ρ in $\mathbb{F}(\mathfrak{n})$, the word $\rho w + z$ is also in $C_{\mathfrak{n}}$. Thus completeness of (P, V) follows from the completeness of $(P_{\mathscr{C}}, V_{\mathscr{C}})$.

Next we argue soundness. Suppose that w is $2\delta_{\mathscr{C}}(\mathfrak{n})$-far from $C_{\mathfrak{n}}$. For every word z in $\mathbb{F}(\mathfrak{n})^{\ell(\mathfrak{n})}$ (not necessarily in $C_{\mathfrak{n}}$), there exists at most one ρ in $\mathbb{F}(\mathfrak{n})$ such that $\rho w + z$ is $\delta_{\mathscr{C}}(\mathfrak{n})$-close to $C_{\mathfrak{n}}$ (see Claim 3.1). The soundness of $(P_{\mathscr{C}}, V_{\mathscr{C}})$ implies that $V_{\mathscr{C}}$ accepts with probability at most $\varepsilon_{\mathscr{C}}(\mathfrak{n})$ if $V_{\mathscr{C}}$ is invoked on a word that is $\delta_{\mathscr{C}}(\mathfrak{n})$-far from $C_{\mathfrak{n}}$. Thus, since V invokes $V_{\mathscr{C}}$ on the word $\rho w + z$, the probability that V accepts is at most $\varepsilon_{\mathscr{C}}(\mathfrak{n}) + \frac{1}{|\mathbb{F}(\mathfrak{n})|}$. □

Claim 7.7. *There exists a straightline simulator S such that, for every $(\mathfrak{n}, w) \in \mathrm{Rel}(\mathscr{C})$ and malicious verifier $\tilde{V}$, the following two random variables are identically distributed*

$$\left(S^{\tilde{V},w}(\mathfrak{n})\ ,\ q_S\right) \quad \textit{and} \quad \left(\mathrm{View}\ \langle P^w(\mathfrak{n}), \tilde{V}^w\rangle\ ,\ q_{\tilde{V}}\right)\ ,$$

where q_S is the number of queries to w made by S and $q_{\tilde{V}}$ is the number of queries to w or to prover messages made by $\tilde{V}$. Moreover, S runs in time $\mathrm{poly}(\mathfrak{n} + \mathsf{q}_{\tilde{V}})$*, where $\mathsf{q}_{\tilde{V}}$ is $\tilde{V}$'s query complexity.*

Proof. We begin by proving perfect zero knowledge via a straightline simulator S_{slow} whose number of queries to w equals $q_{\tilde{V}}$, but runs in time $\text{poly}(\mathtt{tp}(\mathfrak{n}) + q_{\tilde{V}})$. After that, we explain how to modify S_{slow} into another simulator S, with an identical output distribution, that runs in the faster time claimed in the lemma.

The simulator S_{slow}, given straightline access to $\tilde{V}$ and oracle access to w, works as follows:

1. Draw a uniformly random $z_{\text{sim}} \in C_{\mathfrak{n}}$.
2. Whenever $\tilde{V}$ queries w at $\alpha \in D(\mathfrak{n})$, return $w(\alpha)$; whenever $\tilde{V}$ queries z at $\alpha \in D(\mathfrak{n})$, return $z_{\text{sim}}(\alpha)$.
3. Receive $\tilde{\rho}$ from $\tilde{V}$, and draw a uniformly random $w'_{\text{sim}} \| \pi_{\text{sim}} \in L_{\mathfrak{n}}$ conditioned on $w'_{\text{sim}}(\alpha) = \tilde{\rho} w(\alpha) + z_{\text{sim}}(\alpha)$ for every coordinate $\alpha \in D(\mathfrak{n})$ queried in Step 2. (This latter condition requires querying w at α for every coordinate $\alpha \in D(\mathfrak{n})$ queried to z_{sim} in Step 2.)
4. Hereafter: whenever $\tilde{V}$ queries w at $\alpha \in D(\mathfrak{n})$, return $w(\alpha)$; whenever $\tilde{V}$ queries z at $\alpha \in D(\mathfrak{n})$, return $w'_{\text{sim}}(\alpha) - \tilde{\rho} w(\alpha)$. (In either case, a query to w is required.)
5. Tell $\tilde{V}$ that the oracle π has been 'sent'; whenever $\tilde{V}$ queries the i-th entry of π, return the i-th entry of π_{sim}. (Note that $\tilde{V}$ may query w or z before or after learning about π.)
6. Output the view of the simulated $\tilde{V}$.

Note that S_{slow} runs in time $\text{poly}(\mathtt{tp}(\mathfrak{n}) + q_{\tilde{V}})$. Also, S_{slow} makes one query to w for every query to w or z by $\tilde{V}$ (at least provided that $\tilde{V}$'s queries have no duplicates, which we can assume without loss of generality), and zero queries to w for every query to π by $\tilde{V}$. Thus, overall, the number of queries to w by S_{slow} is at most $q_{\tilde{V}}$; clearly, this number of queries can be padded to be equal to $q_{\tilde{V}}$. We now argue that S_{slow}'s output is identically distributed to $\tilde{V}$'s view when interacting with the honest prover P, for z random in $C_{\mathfrak{n}}$.

Claim. $S_{\text{slow}}^{\tilde{V},w} \equiv \text{View}\ \langle P^w(\mathfrak{n}), \tilde{V}^w \rangle$.

Proof. Define the random variable $w' := \tilde{\rho} w + z$, where $\tilde{\rho}$ is chosen by $\tilde{V}$. Observe that there exists a (deterministic) function $v(\cdot)$ such that

$$\text{View}\ \langle P^{w,z}, \tilde{V}^{w,z} \rangle = v(w', w, r) \quad \text{and} \quad S_{\text{slow}}^{\tilde{V},w} = v(w'_{\text{sim}}, w, r)\ ,$$

where the random variable r is $\tilde{V}$'s private randomness. Indeed, (i) the messages sent and received by $\tilde{V}$ are identical to those when interacting with P on w' and w'_{sim}, respectively; (ii) $\tilde{V}$'s queries to w are answered honestly; (iii) $\tilde{V}$'s queries to z are answered by $z = w' - \tilde{\rho} w$ and $z_{\text{sim}} = w'_{\text{sim}} - \tilde{\rho} w$ respectively; (iv) $\tilde{V}$'s queries to π are answered by $P_{\mathscr{C}}(\mathfrak{n}, w')$ and $P_{\mathscr{C}}(\mathfrak{n}, w'_{\text{sim}})$ respectively. We are only left to argue that, for any choice of r, w' and w'_{sim} are identically distributed:

- $w' = \tilde{\rho} w + z$ is uniformly random in $C_{\mathfrak{n}}$, because z is uniformly random in $C_{\mathfrak{n}}$, w is in $C_{\mathfrak{n}}$, and $C_{\mathfrak{n}}$ is linear; and
- w'_{sim} is uniformly random in $C_{\mathfrak{n}}$, because w'_{sim} is sampled at random in $C_{\mathfrak{n}}$ conditioned on $w'_{\text{sim}}(\alpha_i) = z_{\text{sim}}(\alpha_i) + \tilde{\rho} w(\alpha_i)$ for some (adversarial) choice of $\alpha_1, \ldots, \alpha_k$. But z_{sim} is uniformly random in $C_{\mathfrak{n}}$, so the latter condition says that w'_{sim} matches a random codeword on the set of points $\{\alpha_1, \ldots, \alpha_k\}$, giving the claimed distribution for w'_{sim}. □

We explain how to modify S_{slow} so as to reduce the running time to $\text{poly}(\mathfrak{n} + \mathtt{q}_{\tilde{V}})$. The inefficiency of S_{slow} comes from sampling $z_{\text{sim}} \in C_{\mathfrak{n}}$ and $w'_{\text{sim}} \| \pi_{\text{sim}} \in L_{\mathfrak{n}}$, which takes time $\text{poly}(S(\mathfrak{n}) + \ell(\mathfrak{n}))$ and $\text{poly}(\mathtt{tp}_{\mathscr{C}}(\mathfrak{n}))$ respectively, which need not be polynomial in $\mathfrak{n}$. In the following, we show how to not explicitly sample these codewords but, instead, adaptively sample them by relying on constraint detection.

First, note that if $\mathscr{L}$ has constraint detection with a certain efficiency then so does $\mathscr{C}$ with the same efficiency. The theorem's hypothesis says that $\mathscr{L}$ has succinct constraint detection; so both $\mathscr{C}$ and $\mathscr{L}$ have succinct constraint detection. We then invoke Lemma 4.3 to obtain probabilistic polynomial-time algorithms $\mathcal{A}_{\mathscr{C}}, \mathcal{A}_{\mathscr{L}}$ for the code families $\mathscr{C}, \mathscr{L}$ respectively. Using these algorithms, we write the more efficient simulator S, as follows.

Let $D^*(\mathfrak{n})$ be the domain of $L_{\mathfrak{n}}$: this is the disjoint union of $D(\mathfrak{n})$ (the domain of $C_{\mathfrak{n}}$) and $[\mathsf{l}_{\mathscr{C}}(\mathfrak{n})]$ (the domain of $P_{\mathscr{C}}(\mathfrak{n}, C_{\mathfrak{n}})$). The simulator S, given straightline access to $\tilde{V}$ and oracle access to w, works as follows:

1. Let $\mathsf{ans}_{z_{\mathrm{sim}}}$ be a subset of $D(\mathfrak{n}) \times \mathbb{F}(\mathfrak{n})$ that records query-value pairs for z_{sim}; initially, $\mathsf{ans}_{z_{\mathrm{sim}}}$ equals $\varnothing$.
2. Whenever $\tilde{V}$ queries w at $\alpha \in D(\mathfrak{n})$, return $w(\alpha)$; whenever $\tilde{V}$ queries z at $\alpha \in D(\mathfrak{n})$, return $\beta := \mathcal{A}_{\mathscr{C}}(\mathfrak{n}, \mathsf{ans}_{z_{\mathrm{sim}}}, \alpha)$. In the latter case, add (α, β) to $\mathsf{ans}_{z_{\mathrm{sim}}}$.
3. Receive $\tilde{\rho}$ from $\tilde{V}$, and compute $\mathsf{ans} := \{(\alpha, \beta + \tilde{\rho} w(\alpha))\}_{(\alpha,\beta) \in \mathsf{ans}_{z_{\mathrm{sim}}}}$; this subset of $D^*(\mathfrak{n}) \times \mathbb{F}(\mathfrak{n})$ records query-value pairs for $w'_{\mathrm{sim}} \| \pi_{\mathrm{sim}}$.
4. Hereafter: whenever $\tilde{V}$ queries w at $\alpha \in D(\mathfrak{n})$, return $w(\alpha)$; whenever $\tilde{V}$ queries z at $\alpha \in D(\mathfrak{n})$, return $\beta' := \beta - \tilde{\rho} w(\alpha)$ where $\beta := \mathcal{A}_{\mathscr{L}}(\mathfrak{n}, \mathsf{ans}, \alpha)$ and add (α, β) to ans.
5. Tell $\tilde{V}$ that the oracle π has been 'sent'; note that we do not yet commit to any entries in the proof, save for those which are implied by the verifier's previous queries to w. Whenever $\tilde{V}$ queries the i-th location in π, return $\pi_i := \mathcal{A}_{\mathscr{L}}(\mathfrak{n}, \mathsf{ans}, i)$ and add (i, π_i) to ans. (Note that $\tilde{V}$ may query w or z before or after learning about π.)
6. Output the view of the simulated $\tilde{V}$.

Note that S makes the same number of queries to w as S_{slow} does. Also, the number of pairs in $\mathsf{ans}_{z_{\mathrm{sim}}}$ is at most $q_{\tilde{V}}$; ditto for ans. Since the algorithm $\mathcal{A}$ is called at most $q_{\tilde{V}}$ times, the running time of S is $\mathrm{poly}(\mathfrak{n} + \mathsf{q}_{\tilde{V}})$, as required. □

We conclude with a lemma that says that succinct constraint detection is in some sense inherent to the "masking" approach used in Construction 7.5.

Lemma 7.8. *If $(P_{\mathscr{C}}, V_{\mathscr{C}})$ is a linear PCPP such that Construction 7.5 yields (P, V) with perfect zero knowledge, then $(P_{\mathscr{C}}, V_{\mathscr{C}})$ has a constraint detector that runs in probabilistic polynomial time. (In fact, the same statement holds even if the construction yields (P, V) with only statistical zero knowledge.)*

Proof. The linear code $L_{\mathfrak{n}} := \{ w \| P(\mathfrak{n}, w) \}_{w \in C_{\mathfrak{n}}}$ has domain $D^*(\mathfrak{n}) := D(\mathfrak{n}) \sqcup [\mathsf{l}_{\mathscr{C}}(\mathfrak{n})]$, which is the disjoint union of $D(\mathfrak{n})$ (the domain of $C_{\mathfrak{n}}$) and $[\mathsf{l}_{\mathscr{C}}(\mathfrak{n})]$ (the domain of $P(\mathfrak{n}, C_{\mathfrak{n}})$); see Definition 7.2. Let $I \subseteq D^*(\mathfrak{n})$. We need, in probabilistic polynomial time, to output a basis for $(L_{\mathfrak{n}}|_I)^{\perp}$. Construct a malicious verifier $\tilde{V}$ that works as follows:

1. Receive $z \in C_{\mathfrak{n}}$ from P.
2. Send $\rho = 0$ to P.
3. Receive π from P.
4. For each $i \in I$: if $i \in D(\mathfrak{n})$ then query z at i, and if $i \in [\mathsf{l}_{\mathscr{C}}(\mathfrak{n})]$ then query π at i; call the answer β_i.

By PZK, there is a probabilistic polynomial time algorithm S such that the output of $S^{\tilde{V}, w}(\mathfrak{n})$ is identically distributed to $\mathrm{View}\, \langle P^w(\mathfrak{n}), \tilde{V}^w \rangle$, for every $w \in C_{\mathfrak{n}}$. Set w to be the zero codeword, and suppose we run $S^{\tilde{V}, w}(\mathfrak{n})$; this invocation makes S sample answers $(\beta_i)_{i \in I} = (z'(i))_{i \in I}$ for $z' = z \| P_{\mathscr{C}}(\mathfrak{n}, z)$ uniformly random in $L_{\mathfrak{n}}$.

Thus, to perform constraint detection in probabilistic polynomial time, we proceed as follows. We run $S^{\tilde{V}, w}(\mathfrak{n})$ $k > |I|$ times, recording a vector $\vec{\beta}^j = (\beta_i)_{i \in I}$ at the j-th iteration. Let B be the $k \times |I|$ matrix with rows $\vec{\beta}^1, \ldots, \vec{\beta}^k$. Output a basis for the nullspace of B, which we can find in $\mathrm{poly}(\log |\mathbb{F}| + k + |I|)$ time.

We now argue correctness of the above approach. First, for every $u \in \mathbb{F}^I$ such that $\sum_{i \in I} u(i) w'(i) = 0$ for every $w' \in L_{\mathfrak{n}}$, it holds that u is in the nullspace of B, because codewords used to generate B satisfy the same relation. Next, the probability that there exists $u \in \mathbb{F}^I$ in the nullspace of B such that $\sum_{i \in I} u(i) w'(i) \neq 0$ for some $w' \in L_{\mathfrak{n}}$ is at most $1/|\mathbb{F}|^{k-|I|}$. Indeed, for every such u, $\Pr_{z' \leftarrow L_{\mathfrak{n}}}[\sum_{i \in I} u(i) z'(i) = 0] \leq 1/|\mathbb{F}|$ (since $L_{\mathfrak{n}}$ is a linear code), so the probability that u is in the nullspace of B is at most $1/|\mathbb{F}|^k$; we then obtain the claimed probability by a union bound. Overall, the probability that the algorithm answers incorrectly is at most $1/|\mathbb{F}|^{k-|I|}$. □

7.2 Perfect zero knowledge IOPs of proximity for Reed–Solomon codes

We have already proved that the linear code family BS-RS, which consists of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08], has succinct constraint detection. When combined with the results in Section 7.1, we obtain IOPs of Proximity for Reed–Solomon codes, as stated in the corollary below.

Definition 7.9. *We denote by RS^+ the linear code family indexed by tuples $\mathfrak{n} = (\mathbb{F}, L, d)$, where $\mathbb{F}$ is an extension field of $\mathbb{F}_2$ and L is an $\mathbb{F}_2$-linear subspace of $\mathbb{F}$ with $d \leq |L|/8$, and the $\mathfrak{n}$-th code consists of the codewords from the Reed–Solomon code $\mathrm{RS}[\mathbb{F}, L, d]$.*

Theorem 7.10 ([BS08])**.** *For every function $\delta\colon \{0,1\}^* \to (0,1)$, the linear code family RS^+ has PCPs of Proximity with soundness error $1/2$, proximity parameter δ, prover running time (and thus proof length) that is quasilinear in the block length $\ell(\mathfrak{n})$, and verifier running time (and thus query complexity) that is polylogarithmic in $\ell(\mathfrak{n})/\delta(\mathfrak{n})$.*

Corollary 7.11. *For every function $\delta\colon \{0,1\}^* \to (0,1)$, there exists an IOPP system that puts $\mathrm{Rel}(\mathrm{RS}^+)$ in the complexity class*

$$\mathbf{PZK\text{-}IOPP}\left[\begin{array}{lllll}
\text{answer alphabet} & \mathbb{F}(\mathfrak{n}) & & \\
\text{rounds} & \mathsf{k}(\mathfrak{n}) & = & 2 \\
\text{proof length} & \mathsf{l}(\mathfrak{n}) & = & \tilde{O}(\ell(\mathfrak{n})) \\
\text{query complexity} & \mathsf{q}(\mathfrak{n}) & = & \mathrm{polylog}(\ell(\mathfrak{n})/\delta(\mathfrak{n})) \\
\text{soundness error} & \varepsilon(\mathfrak{n}) & = & 1/2 \\
\text{proximity parameter} & \delta(\mathfrak{n}) & & \\
\text{prover time} & \mathsf{tp}(\mathfrak{n}) & = & \tilde{O}(\ell(\mathfrak{n})) \\
\text{verifier time} & \mathsf{tv}(\mathfrak{n}) & = & \mathrm{polylog}(\ell(\mathfrak{n})/\delta(\mathfrak{n})) \\
\text{query bound} & \mathsf{b}(\mathfrak{n}) & = & *
\end{array}\right].$$

Proof. Invoke Theorem 7.3 on the linear code family RS^+ with corresponding BS proximity proofs (Theorem 7.10). Indeed, the concatenation of codewords in RS^+ and proximity proofs yields the family BS-RS, which has succinct constraint detection by Theorem 4.12. (This last step omits a technical, but uninteresting, step: the proximity proofs from Theorem 4.12 consider the case where the degree d *equals* the special value $|L|/8$, rather than being bounded by it; but proximity proofs for smaller degree d are easily obtained from these, as explained in [BS08].) □

When constructing perfect zero knowledge IOPs for $\mathbf{NEXP}$ (Section 8) we shall need perfect zero knowledge IOPs of Proximity not quite for the family RS^+ but for an extension of it that we denote by ERS^+, and for which [BS08] also gives PCPs of proximity. The analogous perfect zero knowledge result follows in a similar way, as explained below.

Definition 7.12. *Given a field $\mathbb{F}$ of characteristic 2, $\mathbb{F}_2$-linear subspaces $H, L \subseteq \mathbb{F}$ with $|H| \leq |L|/8$, and $d_0, d_1 \in \mathbb{N}$ with $d_0, d_1 \leq |L|/8$, we denote by $\mathrm{ERS}^+[\mathbb{F}, L, H, d_0, d_1]$ the linear code consisting of all pairs (w_0, w_1) where $w_0 \in \mathrm{RS}[\mathbb{F}, L, d_0]$, $w_1 \in \mathrm{RS}[\mathbb{F}, L, d_1]$, and $w_1(x) = 0$ for all $x \in H$. We denote by ERS^+ the linear code family indexed by tuples $\mathfrak{n} = (\mathbb{F}, L, d_0, d_1)$ for which the $\mathfrak{n}$-th code is $\mathrm{ERS}^+[\mathbb{F}, L, H, d_0, d_1]$.*

Theorem 7.13 ([BS08])**.** *For every function $\delta\colon \{0,1\}^* \to (0,1)$, the linear code family ERS^+ has PCPs of Proximity with soundness error $1/2$, proximity parameter δ, prover running time (and thus proof length) that is quasilinear in the block length $\ell(\mathfrak{n})$, and verifier running time (and thus query complexity) that is polylogarithmic in $\ell(\mathfrak{n})/\delta(\mathfrak{n})$.*

Proof sketch. A PCP of proximity for a codeword (w_0, w_1) to $\mathrm{ERS}^+[\mathbb{F}, L, H, d_0, d_1]$ consists of (π_0, w_1', π_1), where
- π_0 is a PCP of proximity for w_0 to $\mathrm{RS}[\mathbb{F}, L, d_0]$;
- w_1' is the evaluation of the polynomial obtained by dividing (the polynomial of) w_1 by the zero polynomial of H;
- π_1 is a PCP of proximity for w_1' to $\mathrm{RS}[\mathbb{F}, L, d_1 - |H|]$.

The verifier, which has oracle access to (w_0, w_1) and (π_0, w_1', π_1), checks both PCPs or proximity and then performs a consistency check between w_1 and w_1'. See [BS08] for details. □

Corollary 7.14. *For every function $\delta\colon \{0,1\}^* \to (0,1)$, there exists an IOPP system that puts $\mathrm{Rel}(\mathrm{ERS}^+)$ in the complexity class*

$$\mathbf{PZK\text{-}IOPP}\left[\begin{array}{lllll}
\text{answer alphabet} & \mathbb{F}(\mathfrak{n}) & & \\
\text{rounds} & \mathsf{k}(\mathfrak{n}) & = & 2 \\
\text{proof length} & \mathsf{l}(\mathfrak{n}) & = & \tilde{O}(\ell(\mathfrak{n})) \\
\text{query complexity} & \mathsf{q}(\mathfrak{n}) & = & \mathrm{polylog}(\ell(\mathfrak{n})/\delta(\mathfrak{n})) \\
\text{soundness error} & \varepsilon(\mathfrak{n}) & = & 1/2 \\
\text{proximity parameter} & \delta(\mathfrak{n}) & & \\
\text{prover time} & \mathsf{tp}(\mathfrak{n}) & = & \tilde{O}(\ell(\mathfrak{n})) \\
\text{verifier time} & \mathsf{tv}(\mathfrak{n}) & = & \mathrm{polylog}(\ell(\mathfrak{n})/\delta(\mathfrak{n})) \\
\text{query bound} & \mathsf{b}(\mathfrak{n}) & = & *
\end{array}\right].$$

Proof. Invoke Theorem 7.3 on the linear code family ERS^+ with corresponding BS proximity proofs (Theorem 7.13), which has succinct constraint detection as we now clarify. A codeword (w_0, w_1) has proximity proof (π_0, w_1', π_1), and Theorem 4.12 implies that (w_0, π_0) and (w_1', π_1) have succinct constraint detection. But every coordinate of w_1' is easy to compute from the same coordinate in w_1, and concatenating codewords preserves succinct constraint detection. □

8 Perfect zero knowledge for nondeterministic time

We prove that $\mathbf{NEXP}$ has 2-round IOPs that are perfect zero knowledge against unbounded queries. We do so by constructing a suitable IOP system for $\mathbf{NTIME}(T)$ against query bound b, for each time function T and query bound function b, where the verifier runs in time polylogarithmic in both T and b. Crucially, the simulator runs in time $\mathrm{poly}(\tilde{q} + \log T + \log \mathsf{b})$, where $\tilde{q}$ is the actual number of queries made by the malicious verifier; this exponential improvement over [BCGV16], where the simulator runs in time $\mathrm{poly}(T + \mathsf{b})$, enables us to "go up to $\mathbf{NEXP}$".

Theorem 8.1 (formal statement of Theorem 1.1)**.** *For every constant $d > 0$, time bound function $T\colon \mathbb{N} \to \mathbb{N}$ with $n \leq T(n) \leq 2^{n^d}$, and query bound function $\mathsf{b}\colon \mathbb{N} \to \mathbb{N}$ with $\mathsf{b}(n) \leq 2^{n^d}$, there exists an IOP system (P, V) that makes $\mathbf{NTIME}(T)$ a subset of the complexity class*

$$\mathbf{PZK\text{-}IOP}\left[\begin{array}{lllll} \text{answer alphabet} & \mathbb{F}(n) & = & \mathbb{F}_2 \\ \text{rounds} & \mathsf{k}(n) & = & 2 \\ \text{proof length} & \mathsf{l}(n) & = & \tilde{O}(T(n) + \mathsf{b}(n)) \\ \text{query complexity} & \mathsf{q}(n) & = & \mathrm{polylog}(T(n) + \mathsf{b}(n)) \\ \text{soundness error} & \varepsilon(n) & = & 1/2 \\ \text{prover time} & \mathsf{tp}(n) & = & \mathrm{poly}(n) \cdot \tilde{O}(T(n) + \mathsf{b}(n)) \\ \text{verifier time} & \mathsf{tv}(n) & = & \mathrm{poly}(n + \log(T(n) + \mathsf{b}(n))) \\ \text{query bound} & \mathsf{b}(n) & & \end{array}\right].$$

Moreover, the verifier V is public-coin and non-adaptive.

Our proof is similar to that of [BCGV16], and the only major difference is that [BCGV16]'s simulator *explicitly* samples random codewords, while we rely on succinct constraint detection to do so *implicitly*. Indeed, the reduction from $\mathbf{NTIME}(T)$ generates codewords of size $\tilde{O}(T)$, which means that sampling random codewords of that size is infeasible when T is super-polynomial. We structure our argument in three steps, highlighting the essential components that implicitly underlie [BCGV16]'s 'monolithic' argument; we view this as a conceptual contribution of our work.

Step 1 (Section 8.1). We construct perfect zero knowledge IOPs of Proximity for *linear algebraic constraint satisfaction problems* (LACSPs) [BCGV16], a family of constraint satisfaction problems whose domain and range are linear codes. An instance $\mathtt{x}$ of LACSP is specified by a function g and a pair of codes C_0, C_1; a witness $\mathtt{w}$ for $\mathtt{x}$ is a pair (w_0, w_1) such that $w_0 \in C_0$, $w_1 \in C_1$, and $g(w_0) = w_1$. A natural approach to construct a perfect zero knowledge IOPP for this relation is the following: if we are given a perfect zero knowledge IOP of Proximity for the relation $\mathrm{Rel}(C_0 \times C_1)$, then the verifier can test proximity of $\mathtt{w} = (w_0, w_1)$ to $C_0 \times C_1$ and then sample a random index j and check that $g(w_0)[j] = w_1[j]$. In order for the verifier's strategy to make sense, we require g to (i) satisfy a distance condition with respect to C_0, C_1, namely, that $C_1 \cup g(C_0)$ has large relative distance; (ii) be 'local', which means that computing $g(w_0)[j]$ requires examining only a few indices of w_0; and (iii) be 'evasive', which means that if $\tilde{w}_0$ is close to some $w_0 \in C_0$, then $g(\tilde{w}_0)$ is close to $g(w_0)$. All of this implies that if $(\tilde{w}_0, \tilde{w}_1)$ is far from any valid witness but close to $C_0 \times C_1$, we know that $g(\tilde{w}_0)$ is far from $\tilde{w}_1$, so that examining a random index j gives good soundness.

Step 2 (Section 8.2). We build on the above result to derive perfect zero knowledge IOPs for a subfamily of LACSPs called *randomizable LACSPs* (RLACSPs) [BCGV16]. The key difference between this protocol and the IOP of Proximity described above is that in the "proximity setting", the verifier, and thus also the simulator, has oracle access to the witness, while in the "non-proximity setting" the witness is *sent* to the verifier but the simulator must make do without it; in particular, merely sending the witness (w_0, w_1) is *not* zero knowledge. We thus rely on the randomizability property of RLACSPs to generate witnesses from a t-wise independent distribution, where t is larger than the query bound b. In particular, while the simulator runs in time polynomial in the actual number of queries made by a verifier, it runs in time *polylogarithmic* in t, and thus we can set b to be super-polynomial in order to obtain unbounded-query zero knowledge against polynomial-time verifiers.

Step 3 (Section 8.3). We derive Theorem 8.1 (perfect zero knowledge IOPs for $\mathbf{NTIME}(T)$) by combining: (1) the aforementioned result for RLACSPs; (2) [BCGV16]'s reduction from $\mathbf{NTIME}$ to RLACSPs; (3) a perfect zero knowledge IOP of Proximity for a suitable choice of $C_0 \times C_1$, which we derived in Section 7.2. This last component is the one that makes use of succinct constraint detection, and relies on the technical innovations of our work.

8.1 Perfect zero knowledge IOPs of proximity for LACSPs

A constraint satisfaction problem asks whether, for a given "local" function g, there exists an input w such that $g(w)$ is an accepting output. For example, in the case of 3SAT with n variables and m clauses, the function g maps $\{0,1\}^n$ to $\{0,1\}^m$, and $g(w)$ indicates which clauses are satisfied by $w \in \{0,1\}^n$; hence w yields an accepting output if (and only if) $g(w) = 1^m$. Below we introduce a family of constraint satisfaction problems whose domain and range are linear-algebraic objects, namely, linear codes.

We begin by providing the notion of locality that we use for g, along with a measure of g's "pseudorandomness".

Definition 8.2. *Let $g\colon \Sigma^n \to \Sigma^m$ be a function. We say that g is q-***local** *if for every $j \in [m]$ there exists $I_j \subseteq [n]$ with $|I_j| = q$ such that $g(w)[j]$ (the j-th coordinate of $g(w)$) depends only on $w|_{I_j}$ (the restriction of w to I_j). Moreover, we say that g is s-***evasive** *if for every $I \subseteq [n]$ the probability that I_j intersects I for a uniform $j \in [m]$ is at most $s \cdot \frac{|I|}{n}$.*

For example, if g is a 3SAT formula then g is 3-local because I_j equals the variables appearing in clause j; moreover, g is s-evasive if and only if every variable x_i appears in at most a fraction s/n of the clauses (i.e., the evasiveness property corresponds to the fraction of clauses in which a variable appears). Also, a natural case where g is q-evasive is when the elements of I_j are individually uniform in $[n]$ when j is uniform in $[m]$.

Definition 8.3. *Let $g\colon \Sigma^n \to \Sigma^m$ be a function. We say that g is c-***efficient** *if there is a c-time algorithm that, given j and $w|_{I_j}$, computes the set I_j and value $g(w)[j]$.*

The above definition targets succinctly-described languages. For example, a succinct 3SAT instance is given by a circuit of size S that, on input j, outputs a description of the j-th clause; the definition is then satisfied with $c = O(S)$.

Definition 8.4 (LACSP)**.** *Let $C_0(n), C_1(n)$ be (descriptions of) linear codes over $\mathbb{F}(n)$ with block length $\ell(n)$ and relative distance $\tau(n)$. The promise relation of* **linear algebraic CSPs** *(LACSPs)*

$$(\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{LACSP}})[\mathbb{F}(n), C_0(n), C_1(n), \ell(n), \tau(n), q(n), c(n)]$$

considers instance-witness pairs $(\mathtt{x}, \mathtt{w})$ of the following form.

- *An instance $\mathtt{x}$ is a tuple $(1^n, g)$ where:*
  - *$g\colon \mathbb{F}(n)^{\ell(n)} \to \mathbb{F}(n)^{\ell(n)}$ is $q(n)$-local, $q(n)$-evasive, and $c(n)$-efficient;*
  - *$C_1(n) \cup g(C_0(n))$ has relative distance at least $\tau(n)$ (though may not be a linear space).*
- *A witness $\mathtt{w}$ is a tuple (w_0, w_1) where $w_0, w_1 \in \mathbb{F}(n)^{\ell(n)}$.*

The yes-relation $\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}$ consists of all pairs $(\mathtt{x}, \mathtt{w})$ as above where the instance $\mathtt{x}$ and witness $\mathtt{w}$ jointly satisfy the following: $w_0 \in C_0(n)$, $w_1 \in C_1(n)$, and $g(w_0) = w_1$. (In particular, a witness $\mathtt{w} = (w_0, g(w_0))$ with $w_0 \in C_0(n)$ satisfies $\mathtt{x}$ if and only if $g(w_0) \in C_1(n)$.) The no-language consists of all instances $\mathtt{x}$ as above where $\mathtt{x} \notin \mathrm{Lan}(\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}})$.

Remark 8.5. In [BCGV16] the codes C_0 and C_1 are allowed to have distinct block lengths while, for simplicity, we assume that they have the same block length; this restriction does not change any of their, or our, results.

We are now ready to give perfect zero knowledge IOPs of proximity for LACSPs.

Theorem 8.6. *Suppose that there exists an IOPP system $(\hat{P}, \hat{V})$ that puts $\mathrm{Rel}(C_0 \times C_1)$ in the complexity class*

$$\mathbf{PZK\text{-}IOPP}[\mathbb{F}, \hat{\mathsf{k}}, \hat{\mathsf{l}}, \hat{\mathsf{q}}, \hat{\delta}, \hat{\varepsilon}, \hat{\mathsf{tp}}, \hat{\mathsf{tv}}, *] \ .$$

Then there exists an IOPP system $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ that puts $(\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{LACSP}})[\mathbb{F}, C_0, C_1, \ell, \tau, q, c]$ in the complexity class

$$\mathbf{PZK\text{-}IOPP}\left[\begin{array}{lll} \text{answer alphabet} & \mathbb{F}(n) & \\ \text{rounds} & \mathsf{k}(n) & = \ \hat{\mathsf{k}}(n) \\ \text{proof length} & \mathsf{l}(n) & = \ \hat{\mathsf{l}}(n) \\ \text{query complexity} & \mathsf{q}(n) & = \ \hat{\mathsf{q}}(n) + q(n) + 1 \\ \text{soundness error} & \varepsilon(n) & = \ \max\left\{\hat{\varepsilon}(n)\,,\ 1 - \tau(n) + 2\hat{\delta}(n) \cdot (q(n)+1)\right\} \\ \text{proximity parameter} & \delta(n) & = \ \hat{\delta}(n) \\ \text{prover time} & \mathsf{tp}(n) & = \ \hat{\mathsf{tp}}(n) \\ \text{verifier time} & \mathsf{tv}(n) & = \ \hat{\mathsf{tv}}(n) + c(n) \\ \text{query bound} & \mathsf{b}(n) & = \ * \end{array}\right] \ .$$

Moreover, if $(\hat{P}, \hat{V})$ is non-adaptive (respectively, public-coin) then so is $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$.

Proof. We construct the IOPP system $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ for $\mathscr{R}$, where the prover receives $(\mathtt{x}, \mathtt{w}) = \big((1^n, g), (w_0, w_1)\big)$ as input while the verifier receives $\mathtt{x}$ as input and $\mathtt{w}$ as an oracle, as follows:
1. P_{LACSP} and V_{LACSP} invoke the IOPP system $(\hat{P}, \hat{V})$ to prove that $(w_0, w_1) \in C_0 \times C_1$;
2. V_{LACSP} chooses a random $j \in [\ell]$ and checks that $g(w_0)[j] = w_1[j]$;
3. V_{LACSP} rejects if and only if $\hat{V}$ rejects or the above check fails.

Completeness. If $(\mathtt{x}, \mathtt{w}) \in \mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}$, then (i) $w_0 \in C_0, w_1 \in C_1$, so $\hat{V}$ always accepts, and (ii) $g(w_0) = w_1$ so the consistency check succeeds for every $j \in [\ell]$. We deduce that V_{LACSP} always accepts.

Soundness. Suppose that $\mathtt{x} \in \mathrm{Lan}(\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}) \cup \mathscr{L}^{\mathsf{NO}}_{\mathsf{LACSP}}$ and $\tilde{\mathtt{w}}$ are such that $\Delta(\tilde{\mathtt{w}}, \mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}|_{\mathtt{x}}) \geq \hat{\delta}$. Writing $\tilde{\mathtt{w}} = (\tilde{w}_0, \tilde{w}_1)$, we argue as follows.

- *Case 1:* $\Delta(\tilde{\mathtt{w}}, C_0 \times C_1) \geq \hat{\delta}$. In this case $\hat{V}$ rejects with probability at least $1 - \hat{\varepsilon}$.
- *Case 2:* $\Delta(\tilde{\mathtt{w}}, C_0 \times C_1) < \hat{\delta}$. There exist codewords $w_0 \in C_0$ and $w_1 \in C_1$ such that (w_0, w_1) is δ-close to $(\tilde{w}_0, \tilde{w}_1)$ for $\delta < \hat{\delta}$. By assumption, $\Delta(\tilde{\mathtt{w}}, \mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}|_{\mathtt{x}}) \geq \hat{\delta}$, so in particular (w_0, w_1) cannot be in $\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}|_{\mathtt{x}}$, and $g(w_0) \neq w_1$. Since $C_1 \cup g(C_0)$ has relative distance at least τ, $\Delta(g(w_0), w_1) \geq \tau$. Observe that since C_0 and C_1 have the same block length, $\Delta(\tilde{w}_0, w_0) \leq 2\hat{\delta}$ and $\Delta(\tilde{w}_1, w_1) \leq 2\hat{\delta}$. Thus since g is q-evasive, the probability that the set of coordinates $I := \{i \in [\ell] : w_0[i] \neq \tilde{w}_0[i]\}$ intersects with I_j for random $j \in [\ell]$ is at most $2\hat{\delta}q$, so $\Delta(g(w_0), g(\tilde{w}_0)) \leq 2\hat{\delta}q$. Using the triangle inequality, we deduce that

  $$\Delta(g(\tilde{w}_0), \tilde{w}_1) \geq \tau - 2\hat{\delta}(q+1),$$

  which means the consistency check rejects with probability at least $\tau - 2\hat{\delta}(q+1)$.

It follows that V_{LACSP} accepts with probability at most $\max\{\hat{\varepsilon}, 1 - \tau + 2\hat{\delta}(q+1)\}$.

Perfect zero knowledge. We can choose the simulator S_{LACSP} for $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ to equal any simulator $\hat{S}$ that fulfills the perfect zero knowledge guarantee of $(\hat{P}, \hat{V})$. Indeed, the behavior of P_{LACSP} is the same as $\hat{P}$, and so the view of any malicious verifier $\tilde{V}$ when interacting with P_{LACSP} is identical to its view when interacting with $\hat{P}$. □

8.2 Perfect zero knowledge IOPs for RLACSPs

The above discussion achieves perfect zero knowledge for LACSPs, "up to queries to the witness". We now explain how to simulate these queries as well, without any knowledge of the witness, for a special class of LACSPs called *randomizable LACSPs*. For these, the prover can randomize a given witness $(w_0, g(w_0))$ by sampling a random u' in a t-wise independent subcode C' of C_0, and use the new 'shifted' witness $(w_0 + u', g(w_0 + u'))$ instead of the original one. We now define the notion of randomizable LACSPs, and then show how to construct perfect zero knowledge IOPs for these, against bounded-query verifiers and where the the query bound depends on t.

Definition 8.7 (randomizability)**.** *An instance $\mathtt{x} = (1^n, g)$ is $t(n)$-**randomizable in time** $r(n)$ (with respect to code families $C_0(n), C_1(n)$) if: (i) there exists a $t(n)$-wise independent subcode $C' \subseteq C_0(n)$ such that if $(w_0, g(w_0))$ satisfies $\mathtt{x}$, then, for every w_0' in $C' + w_0 := \{w' + w_0 \mid w' \in C'\}$, the witness $(w_0', g(w_0'))$ also satisfies $\mathtt{x}$; and (ii) one can sample, in time $r(n)$, three uniformly random elements in $C', C_0(n), C_1(n)$ respectively.*

Definition 8.8 (RLACSP)**.** *The promise relation of* **randomizable linear algebraic CSPs** *(RLACSPs) is*

$$(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}})[\mathbb{F}(n), C_0(n), C_1(n), \ell(n), \tau(n), q(n), c(n), t(n), r(n)]$$

where $\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}$ is obtained by restricting $\mathscr{R}_{\mathsf{LACSP}}$ to instances that are t-randomizable in time r, and $\mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}} := \mathscr{L}^{\mathsf{NO}}_{\mathsf{LACSP}}$.

Theorem 8.9. *Suppose that there exists an IOPP system $(\hat{P}, \hat{V})$ that puts $\mathrm{Rel}(C_0 \times C_1)$ in the complexity class*

$$\mathbf{PZK\text{-}IOPP}[\mathbb{F}, \hat{\mathsf{k}}, \hat{\mathsf{l}}, \hat{\mathsf{q}}, \hat{\delta}, \hat{\varepsilon}, \hat{\mathsf{tp}}, \hat{\mathsf{tv}}, *] \ .$$

Then there exists an IOP system $(P_{\mathsf{RLACSP}}, V_{\mathsf{RLACSP}})$ *that puts* $(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}})[\mathbb{F}, C_0, C_1, \ell, \tau, q, c, t, r]$ *(with* c *polynomially bounded) in the complexity class*

$$\textbf{PZK-IOP}\left[\begin{array}{lllll} \text{answer alphabet} & \mathbb{F}(n) & & \\ \text{rounds} & \mathsf{k}(n) & = & \hat{\mathsf{k}}(n) \\ \text{proof length} & \mathsf{l}(n) & = & \hat{\mathsf{l}}(n) + \ell(n) \\ \text{query complexity} & \mathsf{q}(n) & = & \hat{\mathsf{q}}(n) + q(n) + 1 \\ \text{soundness error} & \varepsilon(n) & = & \max\left\{\hat{\varepsilon}(n)\,,\, 1 - \tau(n) + 2 \cdot \hat{\delta}(n) \cdot (q(n)+1)\right\} \\ \text{prover time} & \mathsf{tp}(n) & = & \hat{\mathsf{tp}}(n) + c(n) \cdot \ell(n) + r(n) \\ \text{verifier time} & \mathsf{tv}(n) & = & \hat{\mathsf{tv}}(n) + c(n) \\ \text{query bound} & \mathsf{b}(n) & = & t(n)/q(n) \end{array}\right] .$$

Moreover, if $(\hat{P}, \hat{V})$ *is non-adaptive (respectively, public-coin) then so is* $(P_{\mathsf{RLACSP}}, V_{\mathsf{RLACSP}})$.

Proof. Let $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ be the IOPP system for $\mathscr{R}_{\mathsf{LACSP}}$ guaranteed by Theorem 8.6. We construct the IOP system $(P_{\mathsf{RLACSP}}, V_{\mathsf{RLACSP}})$ for $(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}})$, where the prover receives $(\mathtt{x}, \mathtt{w}) = \big((1^n, g), (w_0, w_1)\big)$ as input while the verifier receives $\mathtt{x}$ as input, as follows:

1. The prover P_{RLACSP} parses the witness $\mathtt{w}$ as $(w_0, w_1) \in C_0 \times C_1$, samples a random $u' \in C'$ (the subcode of C_0 for which t-randomizability holds), sets $w'_0 := u' + w_0$ and $w'_1 := g(w'_0)$, and sends $\mathtt{w}' := (w'_0, w'_1)$ to V_{RLACSP}.

2. In parallel to the above interaction, the prover P_{RLACSP} and verifier V_{RLACSP} invoke the IOPP system $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ on the input $\mathtt{x}$ and new "prover-randomized" witness $\mathtt{w}'$. The verifier V_{RLACSP} accepts if and only if V_{LACSP} does.

The claimed efficiency parameters immediately follow by construction. We now show that $(P_{\mathsf{RLACSP}}, V_{\mathsf{RLACSP}})$ satisfies completeness, soundness, and perfect zero-knowledge.

Completeness. Suppose that $(\mathtt{x}, \mathtt{w}) = \big((1^n, g), (w_0, w_1)\big)$ is in the relation $\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}$, so that $w_0 \in C_0$, $w_1 \in C_1$, and $g(w_0) = w_1$. By randomizability (Definition 8.8), since $w'_0 \in C' + w_0$, we deduce that $\mathtt{w}' = (w'_0, w'_1) = (w'_0, g(w'_0))$ satisfies $\mathtt{x}$, and so $(\mathtt{x}, \mathtt{w}') \in \mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}$. Completeness then follows by the completeness of $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$.

Soundness. Suppose that $\mathtt{x} = (1^n, g)$ is in the language $\mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}} = \mathscr{L}^{\mathsf{NO}}_{\mathsf{LACSP}}$, so that $\mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}|_{\mathtt{x}} = \emptyset$. Regardless of what 'witness' $\mathtt{w}'$ is sent by P_{RLACSP}, it holds that $\Delta(\mathtt{w}', \mathscr{R}^{\mathsf{YES}}_{\mathsf{LACSP}}|_{\mathtt{x}}) = \Delta(\mathtt{w}', \emptyset) = 1 \geq \hat{\delta}$, so that the soundness of $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$ implies that V_{LACSP}, and thus V_{RLACSP}, accepts with probability at most $\max\{\hat{\varepsilon}, 1 - \tau + 2\hat{\delta} \cdot (q+1)\}$.

Perfect zero knowledge. Let $\tilde{V}$ be any verifier that makes at most $\mathsf{b} := t/q$ queries, and let S_{LACSP} be the perfect zero knowledge simulator for $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$. We construct a simulator S_{RLACSP} for $(P_{\mathsf{RLACSP}}, V_{\mathsf{RLACSP}})$ as follows:

$S^{\tilde{V}}_{\mathsf{RLACSP}}(\mathtt{x})$:

1. S_{RLACSP} initializes two empty strings $\hat{w}_0$ and $\hat{w}_1$ which will be partially filled during the simulation.

2. S_{RLACSP} invokes $S^{\tilde{V}, (\hat{w}_0, \hat{w}_1)}_{\mathsf{LACSP}}$, and during the execution answers oracle queries to $(\hat{w}_0, \hat{w}_1)$ in the following way.
   (a) If S_{LACSP} queries $\hat{w}_0$ at a location $j \in [\ell]$: if $\hat{w}_0[j]$ is already defined then return that value; otherwise sample a random $a \in \mathbb{F}(n)$, set $\hat{w}_0[j] := a$, and reply with $\hat{w}_0[j]$.
   (b) If S_{LACSP} queries $\hat{w}_1$ at a location $j \in [\ell]$: if $\hat{w}_1[j]$ is already defined then return that value; otherwise compute the set of indices $I_j \subseteq [\ell]$ that $g(\cdot)_j$ depends on; then 'query' the values of $\hat{w}_0[i]$ for all $i \in I_j$ as in the previous step; then update $\hat{w}_1[j] := g(\hat{w}_0)[j]$ and reply with $\hat{w}_1[j]$.

Observe that S_{RLACSP} runs in time $\mathrm{poly}(|\mathtt{x}| + q_{\tilde{V}} + c)$, where $q_{\tilde{V}}$ denotes the actual number of queries made by $\tilde{V}$ and c is g's efficiency (see Definition 8.3). Since c is polynomially bounded, S_{RLACSP} runs in time $\mathrm{poly}(|\mathtt{x}| + \mathsf{q}_{\tilde{V}})$, as required.

We must show that $\mathrm{View}\ \langle P_{\mathsf{RLACSP}}(\mathtt{x}, \mathtt{w}), \tilde{V}(\mathtt{x})\rangle$ and $S^{\tilde{V}}_{\mathsf{RLACSP}}(\mathtt{x})$ are identically distributed. Recall that $P_{\mathsf{RLACSP}}(\mathtt{x}, \mathtt{w})$ samples $\mathtt{w}'$ and then invokes $P_{\mathsf{LACSP}}(\mathtt{x}, \mathtt{w}')$; viewing $\mathtt{w}'$ as a random variable, we get that $\mathrm{View}\ \langle P_{\mathsf{RLACSP}}(\mathtt{x}, \mathtt{w}), \tilde{V}(\mathtt{x})\rangle \equiv \mathrm{View}\ \langle P_{\mathsf{LACSP}}(\mathtt{x}, \mathtt{w}'), \tilde{V}(\mathtt{x})\rangle$. By $(P_{\mathsf{LACSP}}, V_{\mathsf{LACSP}})$'s perfect zero knowledge guarantee, we also know that

$$\big(\mathrm{View}\ \langle P_{\mathsf{LACSP}}(\mathtt{x}, \mathtt{w}'), \tilde{V}(\mathtt{x})\rangle, q_{\tilde{V}}\big) \equiv \big(S^{\tilde{V}, \mathtt{w}'}_{\mathsf{LACSP}}(\mathtt{x}), q_{S_{\mathsf{LACSP}}}\big)\ .$$

We are left to show that $S^{\tilde{V}, \mathtt{w}'}_{\mathsf{LACSP}}(\mathtt{x}) \equiv S^{\tilde{V}}_{\mathsf{RLACSP}}(\mathtt{x})$.

By the query bound, we know that S_{LACSP} makes at most t/q queries to $\mathtt{w}'$. By construction of S_{RLACSP}, this causes at most t entries in $\hat{w}_0$ to be 'defined', since $|I_j| \leq q$ for all $j \in [\ell]$ (by g's locality); let $E \subseteq [\ell]$ be these entries. Since $w_1 = g(w_0)$, all of the responses to S_{LACSP}'s queries are determined by $w'_0|_E$. While E is itself dependent on w'_0 (as $\tilde{V}$'s queries may be adaptive), this does not affect the distribution of the string $w'_0|_E$ because $|E| \leq t$ and w'_0 is drawn from a t-wise independent distribution. We deduce that there exists a deterministic function $v(\cdot)$ such that S_{LACSP}'s queries to $\mathtt{w}'$ are answered by $v(w'_0|_E)$ in the 'real' execution, and S_{RLACSP} answers the same queries with $v(U)$ where U is uniformly random in $\mathbb{F}^E$. But w'_0 is $|E|$-wise independent, so that $w'_0|_E \equiv U$, and thus $S^{\tilde{V},\mathtt{w}'}_{\mathsf{LACSP}}(\mathtt{x}) \equiv S^{\tilde{V}}_{\mathsf{RLACSP}}(\mathtt{x})$. □

8.3 Putting things together

We are almost ready to prove Theorem 8.1, the main theorem of this section. The last missing piece is a suitable reduction from $\mathbf{NTIME}(T)$ to $\mathscr{R}_{\mathsf{RLACSP}}$, the promise relation of RLACSPs. Below, we state a special case of [BCGV16, Thm. 7.9], which provides the reduction that we need.

Theorem 8.10 ($\mathbf{NTIME} \to \mathscr{R}_{\mathsf{RLACSP}}$)**.** *For every $T, t\colon \mathbb{N} \to \mathbb{N}$, constant $\tau \in (0,1)$, and $\mathscr{R} \in \mathbf{NTIME}(T)$ there exist algorithms $\mathsf{inst}, \mathsf{wit}_1, \mathsf{wit}_2$ satisfying the following conditions:*

- EFFICIENT REDUCTION. *For every instance $\mathtt{x}$, letting $\mathtt{x}' := \mathsf{inst}(\mathtt{x})$:*
  - *if $\mathtt{x} \in \mathrm{Lan}(\mathscr{R})$ then $\mathtt{x}' \in \mathrm{Lan}(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}})$;*
  - *if $\mathtt{x} \notin \mathrm{Lan}(\mathscr{R})$ then $\mathtt{x}' \in \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}}$;*
  - *for every witness $\mathtt{w}$, if $(\mathtt{x}, \mathtt{w}) \in \mathscr{R}$ then $(\mathtt{x}', \mathsf{wit}_1(\mathtt{x}, \mathtt{w})) \in \mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}$;*
  - *for every witness $\mathtt{w}'$, if $(\mathtt{x}', \mathtt{w}') \in \mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}$ then $(\mathtt{x}, \mathsf{wit}_2(\mathtt{x}, \mathtt{w}')) \in \mathscr{R}$.*

  Moreover, inst runs in time $\mathrm{poly}(n + \log(T(n) + t(n)))$ and $\mathsf{wit}_1, \mathsf{wit}_2$ run in time $\mathrm{poly}(n) \cdot \tilde{O}(T(n) + t(n))$.

- RANDOMIZABLE LINEAR ALGEBRAIC CSP. *The promise relation $(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}})$ has the parameters:*

$$(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}}) \left[\begin{array}{lcl} \text{field} & \mathbb{F} & = \mathbb{F}_{2^{\log(T+t)+O(\log\log(T+t))}} \\ \text{first code} & C_0 & \\ \text{second code} & C_1 & \\ \text{block length} & \ell & = \tilde{O}(T+t) \\ \text{relative distance} & \tau & \\ \text{map locality} & q & = \mathrm{polylog}\, T \\ \text{map efficiency} & c & = \mathrm{poly}(n + \log T) \\ \text{randomizability} & t & \\ \text{randomize time} & r & = \tilde{O}(T+t) \end{array}\right].$$

  (The hidden constants depend on the choice of τ; see [BCGV16, Thm. 7.9] for the dependence on τ.)

- ADDITIVE REED–SOLOMON CODES. $\mathrm{Rel}(C_0 \times C_1)$ *is a subfamily of* ERS^+.

Proof of Theorem 8.1. The theorem directly follows by having the prover and verifier reduce the given relation in $\mathbf{NTIME}(T)$ to $(\mathscr{R}^{\mathsf{YES}}_{\mathsf{RLACSP}}, \mathscr{L}^{\mathsf{NO}}_{\mathsf{RLACSP}})$, following Theorem 8.10, and then invoking Theorem 8.9 with the perfect zero knowledge IOP of Proximity for ERS^+ from Corollary 7.14. □

A Prior work on single-prover unconditional zero knowledge

We summarize prior work on *single*-prover proof systems that achieve zero knowledge unconditionally. First, the complexity classes of PZK IPs and SZK IPs are contained in $\mathbf{AM} \cap \mathbf{coAM}$ [For87, AH91], so they do not contain $\mathbf{NP}$ unless the polynomial hierarchy collapses [BHZ87]; thus, IPs have strong limitations. Next, we discuss other single-prover proof systems: PCPs and IPCPs; all prior work for these is about *statistical* zero knowledge (SZK), via simulators that are straightline (which is needed in many of the cryptographic applications explored in these works).

SZK PCP for $\mathbf{NEXP}$. [KPT97] obtain PCPs for $\mathbf{NEXP}$ that are SZK against unbounded queries; the PCP has exponential length, the honest verifier makes a polynomial number of queries, and malicious verifiers can make any polynomial number of queries. Their construction has two steps: (1) transform a given PCP into a new one that is PZK against (several independent copies of) the honest verifier; (2) transform the latter PCP into a new one that is SZK against malicious verifiers. The first step uses secret sharing and builds on techniques of [DFK$^+$92]; the second uses *locking schemes*, which are information-theoretic PCP-analogues of commitment schemes. Subsequent work simplifies the steps: [IW14] use MPC techniques to simplify the first step; and [IMS12, IMSX15] give a simple construction of locking schemes, by obtaining a non-interactive PCP-analogue of [Nao91]'s commitment scheme.

SZK PCP for $\mathbf{NP}$ against unbounded queries. A PCP must have super-polynomial length if it ensures SZK against any polynomial number of malicious queries: if not, a malicious verifier could read the entire PCP, in which case zero knowledge is impossible for non-trivial languages [GO94]. If one allows the prover to be inefficient, then invoking [KPT97]'s result for any language in $\mathbf{NEXP}$, including $\mathbf{NP}$ languages, suffices. Yet, in the case of $\mathbf{NP}$, one can still aim for *oracle efficiency*: the prover outputs a succinct representation of the oracle, i.e., a polynomial-size circuit that, given an index, outputs the value at that index. However, [IMS12, MX13, IMSX15] show that languages with oracle-efficient PCPs that are SZK against unbounded queries are contained in the complexity class of SZP IPs, which is unlikely to contain $\mathbf{NP}$.

SZK PCP for $\mathbf{NP}$ against bounded queries. [KPT97] obtain PCPs for $\mathbf{NP}$ that are SZK against b malicious queries, for a given polynomially-bounded function b. The construction is analogous to the one for $\mathbf{NEXP}$, but with different parameter choices. (The simplifications in [IMS12, IMSX15, IW14] also apply to this case.)

Subsequently, [IW14] consider the case of zero knowledge PCPs *of proximity*; they obtain PCPPs for $\mathbf{NP}$ that are SZK against b malicious queries. Like [KPT97], their construction has two steps: (1) use MPC techniques to transform a given PCPP into a new one that is PZK against (several independent copies of) the honest verifier; (2) use locking schemes to transform the latter PCPP into a new one that is SZK against malicious verifiers.

SZK IPCP for $\mathbf{NP}$ against unbounded queries. For an IPCP to ensure SZK against any polynomial number of queries, the prover must send a PCP with super-polynomial length: if not, a malicious verifier could read the entire PCP, forcing the IPCP model to "collapse" to IP (recall that the complexity class of SZK IPs is unlikely to contain $\mathbf{NP}$). As in the PCP model, one may still aim for oracle efficiency, and this time no limitations apply because a positive result is known: [GIMS10] obtain oracle-efficient IPCPs for $\mathbf{NP}$ that are SZK against unbounded queries. Their construction is analogous to [KPT97]'s, but relies on *interactive locking schemes* in the IPCP model, rather than non-interactive ones in the PCP model; this circumvents the impossibility result for oracle-efficient PCPs.

B Proof of Lemma 4.3

The algorithm $\mathcal{A}$, given $(\mathfrak{n}, S, \alpha)$, where $S = \{(\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)\} \subseteq D(\mathfrak{n}) \times \mathbb{F}(\mathfrak{n})$ and $\alpha \in D(\mathfrak{n})$, works as follows: (1) run $\mathscr{C}$'s constraint detector on input $(\mathfrak{n}, \{\alpha_1, \ldots, \alpha_\ell, \alpha\})$; (2) if the detector outputs an empty basis or a basis $z_1, \ldots, z_d$ where $z_i(\alpha) = 0$ for all i, then output a random element in $\mathbb{F}(\mathfrak{n})$; (3) if the detector outputs some basis element z_j where $z_j(\alpha) \neq 0$, then output $-\sum_{i=1}^{\ell} \frac{z_j(\alpha_i)}{z_j(\alpha)} \beta_i$. The stated time complexity of $\mathcal{A}$ is clear from its construction. We now argue correctness. Define the probability

$$p := \Pr_{w \leftarrow C_{\mathfrak{n}}} \left[w(\alpha) = \beta \;\middle|\; \begin{array}{c} w(\alpha_1) = \beta_1 \\ \vdots \\ w(\alpha_\ell) = \beta_\ell \end{array} \right] .$$

Claim. *(A) If there exist $a_1, \ldots, a_\ell \in \mathbb{F}(\mathfrak{n})$ such that $w(\alpha) = \sum_{i=1}^{\ell} a_i w(\alpha_i)$ for all $w \in C_n$* (Condition A)*, then $p = 1$ if $\beta = \sum_{i=1}^{\ell} a_i \beta_i$ and $p = 0$ otherwise. (B) If no such $a_1, \ldots, a_\ell$ exist, then $p = \frac{1}{|\mathbb{F}(\mathfrak{n})|}$.*

Proof of claim. If Condition A holds, then, for any $w \in C_n$ such that $w(\alpha_1) = \beta_1, \ldots, w(\alpha_\ell) = \beta_\ell$, it holds that $w(\alpha) = \sum_{i=1}^{\ell} a_i w(\alpha_i) = \sum_{i=1}^{\ell} a_i \beta_i$, which proves the first part of the claim.

Next, let $d := \dim(C_n)$ and let $w_1, \ldots, w_d$ be a basis of C_n. Define $\phi_\alpha := (w_1(\alpha), \ldots, w_d(\alpha))$. We argue that Condition A holds if and only if $\phi_\alpha \in \mathrm{span}(\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell})$:

- Suppose that Condition A holds. Then $w_j(\alpha) = \sum_{i=1}^{\ell} a_i w_j(\alpha_i)$ for every $j \in \{1, \ldots, d\}$. Since $w_j(\alpha)$ is the j-th coordinate of ϕ_α, it also holds that $\phi_\alpha = \sum_{i=1}^{\ell} a_i \phi_{\alpha_i}$, so that $\phi_\alpha \in \mathrm{span}(\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell})$.
- Suppose that $\phi_\alpha \in \mathrm{span}(\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell})$. Then there exist $a_1, \ldots, a_\ell$ such that $\phi_\alpha = \sum_{i=1}^{\ell} a_i \phi_{\alpha_i}$. For any $w \in C_n$, we can write $w = \sum_{j=1}^{d} b_j w_j$ (for some b_j's), so that $w(\alpha) = \sum_{j=1}^{d} b_j w_j(\alpha) = \langle w, \phi_\alpha \rangle = \sum_{i=1}^{\ell} a_i \langle w, \phi_{\alpha_i} \rangle = \sum_{i=1}^{\ell} a_i w(\alpha_i)$.

Thus, the negation of Condition A is equivalent to $\phi_\alpha \notin \mathrm{span}(\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell})$, which we now assume to prove the second part of the claim, as follows.

Let $\Phi \in \mathbb{F}(\mathfrak{n})^{\ell \times d}$ be the matrix whose rows are $\phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell}$, and let $w'_1, \ldots, w'_k$ be a basis for Φ's nullspace. Let Φ' be the matrix Φ augmented with the row ϕ_α. Note that $\mathrm{rank}(\Phi') = \mathrm{rank}(\Phi) + 1$, so the nullspace of Φ' has dimension $k - 1$, which implies that there exists $j \in \{1, \ldots, k\}$ such that $\langle w'_j, \phi_\alpha \rangle \neq 0$. Also note that, for every $w \in C_n$ such that $w(\alpha_1) = \beta_1, \ldots, w(\alpha_\ell) = \beta_\ell$ and $r \in \mathbb{F}(\mathfrak{n})$, the codeword $w + r w'_j$ satisfies the same equations as w does. Therefore, if w is drawn uniformly randomly from C_n such that $w(\alpha_1) = \beta_1, \ldots, w(\alpha_\ell) = \beta_\ell$, then $w + r w'_j$ for r uniformly random in $\mathbb{F}(\mathfrak{n})$ is identically distributed to w. We conclude that $\Pr[w(\alpha) = \beta] = \Pr[(w + r w'_j)(\alpha) = \beta] = \Pr[r = \frac{\beta - \langle w, \phi_\alpha \rangle}{\langle w'_j, \phi_\alpha \rangle}] = \frac{1}{|\mathbb{F}(\mathfrak{n})|}$, since $\langle w'_j, \phi_\alpha \rangle \neq 0$. □

By the definition of constraint detection, $a_1, \ldots, a_\ell$ as above exist if and only if there exists z in the space output by the constraint detector such that $z(\alpha) = 1$. If the constraint detector outputs $z_1, \ldots, z_d$ such that $z_i(\alpha) = 0$ for all i, then clearly the space contains no such vector. Otherwise, let j be such that $z_j(\alpha) \neq 0$; then $a_i = -z_j(\alpha_i)/z_j(\alpha)$ for $i = 1, \ldots, \ell$ is a solution. Hence this distribution equals that of $\mathcal{A}$'s output, and moreover fully describes the probability distribution of $w(\alpha)$. The lemma follows.

C Proof of Lemma 4.6

By Claim 4.5, it suffices to show an algorithm that computes a basis of $(C_{\mathfrak{n}}^{\perp})_{\subseteq I}$ in $\mathrm{poly}(|\mathfrak{n}| + |I|)$ time. So consider the algorithm that, on input an index $\mathfrak{n}$ and subset $I \subseteq D(\mathfrak{n})$, works as follows. First, invoke the hypothesis to compute the set W; since vectors are represented sparsely we conclude that $|W|, |\mathrm{supp}(W)| \leq \mathrm{poly}(|\mathfrak{n}| + |I|)$. (Recall that $\mathrm{supp}(W) := \cup_{z \in W} \mathrm{supp}(z)$.) We may assume W is linearly independent; otherwise, make it thus via Gaussian elimination which runs in time $\mathrm{poly}(|W| + |\mathrm{supp}(W)|)$. Similarly, the bound on $|W|$ and $|\mathrm{supp}(W)|$ implies that a basis W' for the subspace $W_{\subseteq I}$ can be found in time $\mathrm{poly}(|\mathfrak{n}| + |I|)$, and we let W' be the output of our algorithm.

To argue correctness it suffices to show that $\mathrm{span}(W') = (C_{\mathfrak{n}}^{\perp})_{\subseteq I}$. We first argue $\mathrm{span}(W') \subseteq (C_{\mathfrak{n}}^{\perp})_{\subseteq I}$, so let $z' \in \mathrm{span}(W')$, which can be represented as $z' = \sum_{\lambda \in \Lambda} a_\lambda \sum_{z \in W} \lambda(z) \cdot z$; note that $z' \in \mathrm{span}(W) \subseteq C_{\mathfrak{n}}^{\perp}$ and $\mathrm{supp}(z') \subseteq \mathrm{supp}(W) = I \cup \overline{I}$. Hence, it suffices to show that $\mathrm{supp}(z') \cap \overline{I} = \emptyset$; but this is true by the choice of Λ, because $M \cdot \lambda = 0$ for every $\lambda \in \Lambda$, so that $\sum_{z \in W} \lambda(w) \cdot z(\alpha) = 0$ for every $\alpha \in \overline{I}$ (by M's definition), so that $z'(\alpha) = \sum_{\lambda \in \Lambda} a_\lambda \sum_{z \in W} \lambda(z) \cdot z(\alpha) = 0$ for every $\alpha \in \overline{I}$, as required.

We next argue that $\mathrm{span}(W') \supseteq (C_{\mathfrak{n}}^{\perp})_{\subseteq I}$, and for this it suffices to show that any $w \in \mathrm{span}(W)$ having representation $w = \sum_{z \in W} a_z \cdot z$ such that $\vec{a} := (a_z)_{z \in W} \notin \mathrm{span}(\Lambda)$ can not be in $(C_{\mathfrak{n}}^{\perp})_{\subseteq I}$. This follows by the definition of Λ, because for any $\vec{a} \notin \mathrm{span}(\Lambda)$ there exists $\alpha \in \overline{I}$ such that $w(\alpha) = \sum_{z \in W} a_z \cdot z(\alpha) \neq 0$, so that $w \notin (C_{\mathfrak{n}}^{\perp})_{\subseteq I}$.

D Proof of Lemma 4.11

For completeness, we give an elementary proof of Lemma 4.11, by simplifying the proof of [Kay10, Thm. 10] for polynomials of the form we require; note that [RS05] and [BW04] also use similar techniques. We first introduce some notation. We consider a polynomial $Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m]$ equivalently as a univariate polynomial of degree less than d in X_1 with coefficients in $\mathbb{F}^{<d}[X_2, \ldots, X_m]$, and let $\partial_1^j Q$ be the coefficient of X_1^j in this representation. Define $\partial_1^j \vec{Q} := (\partial_1^j Q_1, \ldots, \partial_1^j Q_\ell)$. In general, given an arbitrary arithmetic circuit representing a polynomial Q, it is not clear how to efficiently compute a circuit representing $\partial_1^j Q$, because Q may have exponentially many monomials. Nevertheless, for circuits of the required form, this computation is trivial.

Claim. *Let $\vec{Q} := (Q_1, \ldots, Q_\ell)$ be a vector of polynomials in $\mathbb{F}^{<d}[X_1, \ldots, X_m]$. If $d \le |\mathbb{F}|$ then $\vec{Q}^\perp = \bigcap_{j=0}^{d-1} (\partial_1^j \vec{Q})^\perp$.*

Proof. When $d \le |\mathbb{F}|$, $Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m] \equiv 0$ if and only if all of its coefficients are zero when written as a formal sum. Then one direction of the set equality follows straightforwardly from the linearity of ∂_1^j, namely, $\vec{Q}^\perp \subseteq \bigcap_{j=0}^{d-1} (\partial_1^j \vec{Q})^\perp$. For the other direction, we argue as follows. Fix some $(a_1, \ldots, a_\ell) \in \bigcap_{j=0}^{d-1} (\partial_1^j \vec{Q})^\perp$ and let $T := \sum_{k=1}^{\ell} a_k Q_k$; we have that $\partial_1^j T \equiv 0$ for all $j \in \{0, \ldots, d-1\}$, by linearity. But $T = \sum_{j=0}^{d-1} (\partial_1^j T) X_1^j$ by definition, so $T \equiv 0$, and thus $(a_1, \ldots, a_\ell) \in \vec{Q}^\perp$. □

Thus to compute a basis of $\vec{Q}^\perp$ it suffices to compute the intersection of the bases of $(\partial_1^j \vec{Q})^\perp$ for all $j \in \{0, \ldots, d-1\}$. The naive approach yields an exponential-time algorithm since we reduce the problem to d subproblems of roughly the same size. Observe, however, that for Q_k of the specified form,

$$\partial_1^j Q_k = c_{k,j} T_k \quad \text{where } T_k := \left(\prod_{i=2}^{m} Q_{k,i}(X_i) \right) ,$$

for constants $c_{k,j}$ computable in time $\mathrm{poly}(s)$. Let $\vec{T} := (T_1, \ldots, T_\ell)$ and let $T^\perp \in \mathbb{F}^{\ell \times b}$ be a basis for $\vec{T}^\perp$; note that $b \le \ell$. Let $\vec{a} := (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell$, and observe that for each j, $\sum_{k=1}^{\ell} a_k \partial_1^j Q_k \equiv 0$ if and only if $\sum_{k=1}^{\ell} a_k c_{k,j} T_k \equiv 0$, or equivalently, $(a_1 c_{1,j}, \ldots, a_\ell c_{\ell,j}) \in \vec{T}^\perp$. Hence $(a_1, \ldots, a_\ell) \in \bigcap_{j=0}^{d} (\partial_1^j \vec{Q})^\perp$ if and only if for each j there exists $\vec{v_j} \in \mathbb{F}^b$ such that $T^\perp \vec{v_j} = (a_1 c_{1,j}, \ldots, a_\ell c_{\ell,j})$. This is a system of linear equations in $\vec{a}, \vec{v_0}, \ldots, \vec{v_{d-1}}$ of size $\mathrm{poly}(\ell + d + b)$, and hence we can compute a basis for its solution space in time $\mathrm{poly}(\log |\mathbb{F}| + d + \ell + b)$. Restricting this basis to $\vec{a}$ yields a basis for $\vec{Q}^\perp$.

If $Q_1, \ldots, Q_n$ are univariate then we can easily determine a basis for $\vec{Q}^\perp$ in deterministic polynomial time (by Gaussian elimination). Otherwise, if the Q_i are m-variate, we use the procedure above to reduce computing $\vec{Q}^\perp$ to computing $\vec{T}^\perp$ for some $\vec{T} = (T_1, \ldots, T_\ell)$ where the T_i are $(m-1)$-variate. This algorithm terminates in time $\mathrm{poly}(\log |\mathbb{F}| + m + d + s + \ell)$.

E Proof of Claim 4.23

First we show that $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp}) \subseteq (C^{\perp})_{\subseteq(\cup_{j\in J}\tilde{D}_j)}$. For every $j \in J$ and $z \in \tilde{C}_j^{\perp}$, it holds that $\mathrm{supp}(z) \subseteq \tilde{D}_j$; therefore, for every $z \in \mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp})$, it holds that $\mathrm{supp}(z) \subseteq \cup_{j\in J}\tilde{D}_j$; thus it is suffices to show that, for every $z \in \cup_{j\in J}\tilde{C}_j^{\perp}$ and $w \in C$, it holds that $\langle w, z\rangle = 0$. But this holds because for every $z \in \cup_{j\in J}\tilde{C}_j^{\perp}$ there exists $j \in J$ such that $z \in \tilde{C}_j^{\perp}$ and $C|_{\tilde{D}_j} = \tilde{C}_j$ so that $\langle w, z\rangle = \langle w|_{\tilde{D}_j}, z\rangle = 0$, as required.

Next we show that $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp}) \supseteq (C^{\perp})_{\subseteq(\cup_{j\in J}\tilde{D}_j)}$, which is equivalent to $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp}) \supseteq (C|_{\cup_{j\in J}\tilde{D}_j})^{\perp}$ by Claim 4.5. Recall that for any two linear spaces U, V it holds that $U \subseteq V$ if and only if $U^{\perp} \supseteq V^{\perp}$, thus it is sufficient to show that $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp})^{\perp} \subseteq C|_{\cup_{j\in J}\tilde{D}_j}$, i.e., that every $w \in \mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp})^{\perp}$ can be extended to $w' \in C$. This latter statement holds because $\mathrm{span}(\cup_{j\in J}\tilde{C}_j^{\perp})^{\perp}|_{\tilde{D}_j} \subseteq (\tilde{C}_j^{\perp})^{\perp} = \tilde{C}_j$ for every $j \in J$, and thus $w|_{\tilde{D}_j} \in \tilde{C}_j$. Recalling $|J| \leq \kappa$ implies, by Definition 4.15, that w can be extended to a codeword $w' \in C$, as claimed.

F Definition of the linear code family BS-RS

In this section we define the linear code family BS-RS, which consists of evaluations of univariate polynomials concatenated with corresponding BS proximity proofs [BS08]. The definition is quite technical, and we refer the interested reader to [BS08] for a discussion of why it enables proximity testing. We begin with notation used later.

Definition F.1. *Given a field $\mathbb{F}$, a subfield $\mathbb{K} \subseteq \mathbb{F}$, a $\mathbb{K}$-linear space $L \subseteq \mathbb{F}$ with a basis $(b_1, b_2, \ldots, b_\ell)$, a positive integer μ, and a positive integer $k > 2\mu$, we make the following definitions.*

- *Four subspaces of L and a subset of L:*

$$\begin{aligned}
L_0[\mathbb{K},\mathbb{F},L,\mu] &:= \mathrm{span}_{\mathbb{K}}(b_1, b_2, \ldots, b_{\lfloor \ell/2 \rfloor}) \\
L_0'[\mathbb{K},\mathbb{F},L,\mu] &:= \mathrm{span}_{\mathbb{K}}(b_1, b_2, \ldots, b_{\lfloor \ell/2 \rfloor+\mu-1}) \\
L_1[\mathbb{K},\mathbb{F},L,\mu] &:= \mathrm{span}_{\mathbb{K}}(b_{\lfloor \ell/2 \rfloor+1}, \ldots, b_{\ell}) \\
\forall\, \beta \in L_1[\mathbb{K},\mathbb{F},L,\mu]\,,\ L_\beta[\mathbb{K},\mathbb{F},L,\mu] &:= \mathrm{span}_{\mathbb{K}}(b_1, b_2, \ldots, b_{\lfloor \ell/2 \rfloor+\mu-1}, \beta') \\
\forall\, \beta \in L_1[\mathbb{K},\mathbb{F},L,\mu]\,,\ R_\beta[\mathbb{K},\mathbb{F},L,\mu] &:= L_\beta \setminus (L_0 + \beta)
\end{aligned}$$

  where $\beta' := b_{\lfloor \ell/2 \rfloor+\mu}$ if $\beta \in L_0'$ and $\beta' := \beta$ otherwise.

- *The vanishing polynomial of L_0: $Z_{L_0}[\mathbb{K},\mathbb{F},L,\mu](X) := \prod_{\alpha \in L_0}(X - \alpha)$.*

- *The following domains:*

$$\begin{aligned}
D_{\mathrm{bi}}[\mathbb{K},\mathbb{F},L,\mu] &:= \{(\alpha, Z_{L_0}(\beta)) : \beta \in L_1, \alpha \in L_\beta\} \\
D_{\mathrm{pf}}[\mathbb{K},\mathbb{F},L,\mu] &:= \{(\alpha, Z_{L_0}(\beta)) : \beta \in L_1, \alpha \in R_\beta\} \\
D_{\square}[\mathbb{K},\mathbb{F},L,\mu] &:= (\{\mathrm{rs}\} \times L) \sqcup (\{\mathrm{px}\} \times D_{\mathrm{pf}})
\end{aligned}$$

  where we use the symbols 'rs' and 'px' to distinguish different parts of the disjoint union.

- *The bijection $\phi[\mathbb{K},\mathbb{F},L,\mu] \colon D_{\mathrm{bi}} \to D_{\square}$ is defined by $\phi(\alpha,\beta) := \begin{cases} (\mathrm{px}, (\alpha,\beta)) & (\alpha,\beta) \in D_{\mathrm{pf}} \\ (\mathrm{rs}, \alpha) & \text{otherwise} \end{cases}$.*

- *Given $w \in \mathbb{F}^{D_{\square}[\mathbb{K},\mathbb{F},L,\mu]}$, the bivariate function $f_w \colon D_{\mathrm{bi}}[\mathbb{K},\mathbb{F},L,\mu] \to \mathbb{F}$ is defined by $f_w(\alpha,\beta) := w(\phi(\alpha,\beta))$.*

- *The fractional degree $\rho[\mathbb{K},\mathbb{F},\mu] := |\mathbb{K}|^{-\mu}$.*

- *The domain $D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]$ implied by the recursion below:*
  - *if $\dim(L) \le k$ then $D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k] := D_{\mathrm{pf}}[\mathbb{K},\mathbb{F},L,\mu]$;*
  - *if $\dim(L) > k$ then*

$$\begin{aligned}
D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k] := D_{\mathrm{pf}}[\mathbb{K},\mathbb{F},L,\mu] &\bigsqcup \Big(\sqcup_{\alpha \in L_0'} \{(\mathrm{col},\alpha)\} \times D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k] \Big) \\
&\bigsqcup \Big(\sqcup_{\beta \in L_1} \{(\mathrm{row},\beta)\} \times D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k] \Big)\ .
\end{aligned}$$

    where we use the symbols 'col' and 'row' to distinguish different parts of the disjoint union.

- *The domain $D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k] := (\{\mathrm{rs}\} \times L) \sqcup \big(\{\mathrm{px}\} \times D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]\big)$.*

- *Given $\alpha \in L_0'$, the embedding $\phi_{\mathrm{col},\alpha} \colon D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k] \hookrightarrow D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]$ is defined by*

$$\phi_{\mathrm{col},\alpha}(x) := \begin{cases} (\mathrm{px}, ((\mathrm{col},\alpha), x)) & x \in \{\mathrm{px}\} \times D_{\mathrm{px}}^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k] \\ \phi(\alpha,\beta) & x = (\{\mathrm{rs}\}, Z_{L_0}(\beta)) \end{cases}$$

  We denote by $D_{\mathrm{col},\alpha}$ the image of $\phi_{\mathrm{col},\alpha}$.

- *Given $\beta \in L_1$, the embedding $\phi_{\mathrm{row},\beta} \colon D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k] \hookrightarrow D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]$ is defined by*

$$\phi_{\mathrm{row},\beta}(x) := \begin{cases} (\mathrm{px},((\mathrm{row},\beta),x)) & x \in \{\mathrm{px}\} \times D^{\text{BS-RS}}_{\mathrm{px}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k] \\ \phi(\alpha,\beta) & x = (\{\mathrm{rs}\},\alpha) \end{cases}$$

  We denote by $D_{\mathrm{row},\beta}$ the image of $\phi_{\mathrm{row},\beta}$.

- *Given $\alpha \in L'_0$, $\psi_{\mathrm{col},\alpha} \colon \mathbb{F}^{D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]} \to \mathbb{F}^{D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]}$ is the projection of $D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]$ on $D_{\mathrm{col},\alpha}$ with indices renamed to elements of $D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$. Formally, $\psi_{\mathrm{col},\alpha}(w) = w'$ if and only if $w'(\phi_{\mathrm{col},\alpha}(x)) = w(x)$ for all $x \in D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$.*

- *Given $\beta \in L_1$, $\psi_{\mathrm{row},\beta} \colon \mathbb{F}^{D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]} \to \mathbb{F}^{D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k]}$ is the projection of $D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]$ on $D_{\mathrm{row},\beta}$ with indices renamed to elements of $D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k]$. Formally, $\psi_{\mathrm{row},\beta}(w) = w'$ if and only if $w'(\phi_{\mathrm{row},\beta}(x)) = w(x)$ for all $x \in D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L_\beta,\mu,k]$.*

The following definition considers a code that extends the evaluation of a univariate polynomial with a bivariate function that represents the polynomial over a specially-chosen set.

Definition F.2 ($\mathrm{RS}_\square$)**.** *Given a field $\mathbb{F}$, a subfield $\mathbb{K} \subseteq \mathbb{F}$, a $\mathbb{K}$-linear space $L \subseteq \mathbb{F}$, and a positive integer μ, the code $\mathrm{RS}_\square[\mathbb{K},\mathbb{F},L,\mu]$ consists of all $w \in \mathbb{F}^{D_\square[\mathbb{K},\mathbb{F},L,\mu]}$ such that $f_w \colon D_{\mathrm{bi}} \to \mathbb{F}$ is an evaluation of a low degree polynomial: there exists a polynomial $g \in \mathbb{F}[X,Y]$ such that: (i) $\deg_X(g) < |L_0|$, (ii) $\deg_Y(g) < |L_1| \cdot \rho[\mathbb{K},\mathbb{F},\mu]$, (iii) $g|_{D_{\mathrm{bi}}} = f_w$.*

Ben-Sasson and Sudhan [BS08] show that:

- $v \in \mathrm{RS}[\mathbb{F},L,|L| \cdot \rho]$ if and only if there exists $w \in \mathrm{RS}_\square[\mathbb{K},\mathbb{F},L,\mu]$ such that $w|_{\{\mathrm{rs}\} \times L} = v$;
- $w \in \mathrm{RS}_\square[\mathbb{K},\mathbb{F},L,\mu]$ if and only if
  - for every $\alpha \in L'_0$, $f_w|_{\{\alpha\} \times Z_{L_0}(L_1)} \in \mathrm{RS}[\mathbb{F},Z_{L_0}(L_1),|L_1| \cdot \rho]$ (with the standard mapping between domains) and
  - for every $\beta \in L_1$, $f_w|_{L_\beta \times \{Z_{L_0}(\beta)\}} \in \mathrm{RS}[\mathbb{F},L_\beta,|L_0|]$ (with the standard mapping between domains).

The above equivalences illustrate the 'quadratic reduction' from testing that $w \in \mathbb{F}^L$ is a codeword of $\mathrm{RS}[\mathbb{F},L,|L| \cdot \rho]$ to a set of $\Theta(\sqrt{|L|})$ problems of testing membership in codes of the form $\mathrm{RS}[\mathbb{F},L',d']$ with $|L'|,d' = \Theta(\sqrt{|L|})$.

The code from Definition F.2 corresponds to one step of the recursive construction of [BS08]. We now build on that definition, and recursively define the linear code family BS-RS.

Definition F.3 (BS-RS)**.** *Given a field $\mathbb{F}$, a subfield $\mathbb{K} \subseteq \mathbb{F}$, a $\mathbb{K}$-linear space $L \subseteq \mathbb{F}$, a positive integer μ, and a positive integer $k > 2\mu$, the code $\text{BS-RS}[\mathbb{K},\mathbb{F},L,\mu,k]$ consists of all words $w \in \mathbb{F}^{D^{\text{BS-RS}}[\mathbb{K},\mathbb{F},L,\mu,k]}$ satisfying the following. If $\dim(L) \leq k$ then $w \in \mathrm{RS}_\square[\mathbb{K},\mathbb{F},L,\mu]$. If $\dim(L) > k$ the following holds: (1) for every $\alpha \in L'_0$ there exists $w_\alpha \in \text{BS-RS}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$ such that $w_\alpha(\phi_{\mathrm{col},\alpha}(x)) = w(x)$ for every $x \in D[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$; (2) for every $\beta \in L_1$ there exists $w_\beta \in \text{BS-RS}[\mathbb{K},\mathbb{F},L_\beta,\mu,k]$ such that $w_\beta(\phi_{\mathrm{row},\beta}(x)) = w(x)$ for every $x \in D[\mathbb{K},\mathbb{F},L_\beta,\mu,k]$.*

We conclude this section with two claims about BS-RS that we use in later sections. We omit the proof of the first claim (and refer the interested reader to [BS08]), and prove the second claim based on the first one.

Claim F.4. *For every codeword $w \in \mathrm{RS}[\mathbb{F},L,|L| \cdot \rho]$, positive integer μ, and positive integer $k > 2\mu$, there exists a unique π_w such that $w \circ \pi_w \in \text{BS-RS}[\mathbb{K},\mathbb{F},L,\mu,k]$.*

Claim F.5. *The following two statements hold for the code $\text{BS-RS}[\mathbb{K},\mathbb{F},L,\mu,k]$:*

- *for every $\alpha \in L'_0$ and $w' \in \text{BS-RS}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$ there exists $w \in \text{BS-RS}[\mathbb{K},\mathbb{F},L,\mu,k]$ such that $\psi_{\mathrm{col},\alpha}(w) = w'$;*
- *for every $\beta \in L_1$ and $w' \in \text{BS-RS}[\mathbb{K},\mathbb{F},L_\beta,\mu,k]$ there exists $w \in \text{BS-RS}[\mathbb{K},\mathbb{F},L,\mu,k]$ such that $\psi_{\mathrm{row},\beta}(w) = w'$.*

Proof. The proofs for the two statements are similar, so we only give the proof for the first statement. Let $w' \in \text{BS-RS}[\mathbb{K},\mathbb{F},Z_{L_0}(L_1),\mu,k]$, and define $w_{\mathrm{rs}} := w'|_{\{\mathrm{rs}\} \times Z_{L_0}(L_1)}$; observe that w_{rs} in $\mathrm{RS}[\mathbb{F},Z_{L_0}(L_1),|L_1| \cdot \rho]$. By Claim F.4, w' is uniquely determined by w_{rs}, thus it suffices to show that there exists $w_\square \in \mathrm{RS}_\square[\mathbb{K},\mathbb{F},L,\mu]$ such that $f_{w_\square}|_{\{\alpha\} \times Z_{L_0}(L_1)} = w_{\mathrm{rs}}$. By definition of $\mathrm{RS}_\square$, it suffices to show that there exists a bivariate polynomial $g \in \mathbb{F}[X,Y]$ such that: (i) $\deg_X(g) < |L_0|$, (ii) $\deg_Y(g) < |L_1| \cdot \rho$, (iii) $g|_{\{\alpha\} \times Z_{L_0}(L_1)} = w_{\mathrm{rs}} \in \mathrm{RS}[\mathbb{F},Z_{L_0}(L_1),|L_1| \cdot \rho]$. The existence of such g follows by considering a suitable interpolating set (see, e.g., Appendix H). □

G Proof of Lemma 4.27

In this section we prove Lemma 4.27. In Appendix G.1 we define the recursive cover and prove its combinatorial properties; in Appendix G.2 we prove that a spanning set for the duals of codes in this cover can be computed efficiently; in Appendix G.3, we put these together to conclude the proof.

G.1 The recursive cover and its combinatorial properties

We define a recursive cover for BS-RS and then prove certain combinatorial properties for it. The definition relies on the definition of another cover, which we now introduce.

Definition G.1. *The* **native cover** $S[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *of* $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *is defined as follows:*

- *if* $\dim(L) \leq k$ *then the cover contains only the trivial view* $(D^{\text{BS-RS}}[\mathbb{K}, \mathbb{F}, L, \mu, k], \text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k])$*;*
- *if* $\dim(L) > k$ *then the cover contains*
  - *the view* $(\text{BS-RS}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k], D_{\text{col},\alpha})$ *for every* $\alpha \in L'_0$*, and*
  - *the view* $(\text{BS-RS}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k], D_{\text{row},\beta})$ *for every* $\beta \in L_1$.

We now prove that the native cover is indeed a cover.

Claim G.2. *The native cover of* $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *is a code cover (see Definition 4.13).*

Proof. From Claim F.5 we know that:
- for every $\alpha \in L'_0$, the restriction of $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ to $D_{\text{col},\alpha}$ equals $\text{BS-RS}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$;
- for every $\beta \in L_1$, the restriction of $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ to $D_{\text{row},\beta}$ equals $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k]$.

Therefore, it suffices to show that $D_\square \subseteq (\cup_{\alpha \in L'_0} D_{\text{col},\alpha}) \cup (\cup_{\beta \in L_1} D_{\text{row},\beta})$. So let x be an index in $D_\square$.
- If there exists $\alpha \in L'_0$ such that $x \in \{\text{px}\} \times \{(\text{col}, \alpha)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$, then $x \in D_{\text{col},\alpha}$.
- If there exists $\beta \in L_1$ such that $x \in \{\text{px}\} \times \{(\text{row}, \beta)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k]$, then $x \in D_{\text{row},\beta}$.
- If $x \in \{\text{px}\} \times D_{\text{pf}}[\mathbb{K}, \mathbb{F}, L, \mu]$, then there exist $\beta \in L_1$ and $\alpha \in R_\beta$ such that $x = (\text{px}, (\alpha, Z_{L_0}(\beta)))$, so $x \in D_{\text{row},\beta}$.
- If $x \in \{\text{rs}\} \times L$, then there exist $\beta \in L_1$ and $\alpha \in L_\beta$ such that $\phi[\mathbb{K}, \mathbb{F}, L, \mu](\alpha, Z_{L_0}(\beta)) = x$, so $x \in D_{\text{row},\beta}$. □

The recursive cover of BS-RS is recursively defined based on the native cover of BS-RS.

Definition G.3. *The recursive cover* $T[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *of* $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *is the tree of depth* $\lfloor \log \dim(L) - \log(k) \rfloor$ *where, for every non-leaf vertex* v *labeled by* $(\tilde{D}, \text{BS-RS}[\mathbb{K}, \mathbb{F}, \tilde{L}, \mu, k])$*, the vertex* v *has* $|S[\mathbb{K}, \mathbb{F}, \tilde{L}, \mu, k]|$ *successors, all labeled by elements of* $S[\mathbb{K}, \mathbb{F}, \tilde{L}, \mu, k]$ *with the natural embedding of their domains into* $\tilde{D}$*.*

Claim G.4. *The recursive cover of* $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ *is* 1*-intersecting (see Definition 4.18).*

Proof. We must show that for every two disconnected vertices u, v it holds that $|\tilde{D}_u \cap \tilde{D}_v| \leq 1$. It suffices to do so for every two distinct *siblings* u, v, because if a is an ancestor of b then $\tilde{D}_a$ contains $\tilde{D}_b$. Hence, we only need to show that for every two distinct views $(\tilde{D}, \tilde{C}), (\tilde{D}', \tilde{C}')$ in the native cover $S[\mathbb{K}, \mathbb{F}, L, \mu, k]$, it holds that $|\tilde{D} \cap \tilde{D}'| \leq 1$. First we observe that for every $\alpha_1 \neq \alpha_2 \in L'_0$ and $\beta_1 \neq \beta_2 \in L_1$, the following sets are disjoint by definition:
- $\{\text{px}\} \times \{(\text{col}, \alpha_1)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$,
- $\{\text{px}\} \times \{(\text{col}, \alpha_2)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$,
- $\{\text{px}\} \times \{(\text{row}, \beta_1)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, L_{\beta_1}, \mu, k]$,
- $\{\text{px}\} \times \{(\text{row}, \beta_2)\} \times D^{\text{BS-RS}}_{\text{pf}}[\mathbb{K}, \mathbb{F}, L_{\beta_2}, \mu, k]$.

Thus it is enough to show that:
- Any two columns are distinct: $\phi(\alpha_1, \beta_1) \neq \phi(\alpha_2, \beta_2)$ for every $\alpha_1 \neq \alpha_2 \in L'_0$ and $\beta_1, \beta_2 \in Z_{L_0}(L_1)$.
- Any two rows are distinct: $\phi(\alpha_1, \beta_1) \neq \phi(\alpha_2, \beta_2)$ for every $\beta_1 \neq \beta_2 \in Z_{L_0}(L_1)$, $\alpha_1 \in L_{\beta_1}$, and $\alpha_2 \in L_{\beta_2}$.
- The intersection of any row and column has at most one element: $\phi(\alpha, \beta') \neq \phi(\alpha', \beta)$ for every $\alpha, \alpha' \in L'_0$ and $\beta, \beta' \in Z_{L_0}(L_1)$ with $(\alpha', \beta') \neq (\alpha, \beta)$.

But all the above follow from the fact that $\phi[\mathbb{K}, \mathbb{F}, L, \mu]$ is a bijection and, thus, an injection. □

The next claim establishes a connection between the depth of a vertex v in the recursive cover and the independence of the cover T_v of the code $\tilde{C}_v$.

Claim G.5. *For every vertex v in $\mathrm{layer}(T, d)$, the cover T_v is $(|\mathbb{K}|^{\dim(L)\cdot 2^{-d-1}-\mu-2})$-independent. In particular, by assignment, it holds that, for every positive integer m and every non-leaf vertex v in $T[\mathbb{K}, \mathbb{F}, L, \mu, k]$ with depth less than $\log_2 \dim(L) - \log_2(\log_{|\mathbb{K}|} m + \mu + 2) - 1$, the cover T_v is m-independent.*

The proof of the above claim directly follows from Claim G.7 and Claim G.6, stated and proved below. The first of these two claims connects the depth of a vertex v and the dimension of a space L_v such that $\tilde{C}_v = \text{BS-RS}[\mathbb{F}, \mathbb{K}, L_v, \mu, k]$ (this claim is used separately also for establishing computational properties in in Appendix G.2).

Claim G.6. *If $v \in \mathrm{layer}(T[\mathbb{K}, \mathbb{F}, L, \mu, k], d)$ then $\tilde{C}_v = \text{BS-RS}[\mathbb{K}, \mathbb{F}, \tilde{L}, \mu, k]$ for some $\tilde{L}$ such that*

$$\dim(L) \cdot 2^{-d} \leq \dim(\tilde{L}) \leq \dim(L) \cdot 2^{-d} + 2\mu \ .$$

Proof. The proof is by induction on d. The base case $d = 0$ follows directly from the definition; so we now assume the claim for $d-1$ and prove it for d. Let $v \in \mathrm{layer}(T, d)$ be a vertex of depth d, and let $u \in \mathrm{layer}(T, d-1)$ be v's predecessor. By the inductive assumption, $\tilde{C}_u = \text{BS-RS}[\mathbb{K}, \mathbb{F}, L_u, \mu, k]$ for some L_u such that $\dim(L) \cdot 2^{-(d-1)} \leq \dim(L_u) \leq \dim(L) \cdot 2^{-(d-1)} + 2\mu$.

First we argue that T_u is not the trivial (singleton) cover. For this, it suffices to show that $\dim(L_u) > k$. But this follows from the inductive assumption, since $\mathrm{depth}(T, u) < \lfloor \log \dim(L) - \log(k) \rfloor$, so that $\dim(L_u) \geq \dim(L) \cdot 2^{-(\lfloor \log \dim(L) - \log(k) \rfloor - 1)} \geq 2k$.

Recall $\tilde{C}_v = \text{BS-RS}[\mathbb{K}, \mathbb{F}, L_v, \mu, k]$ for some space L_v; we are thus left to show that $\dim(L_u) \cdot 2^{-1} \leq \dim(L_v) \leq \dim(L_u) \cdot 2^{-1} + \mu$. We do so by giving two cases, based on the form of L_v: (a) if $L_v = Z_{L_0[\mathbb{K}, \mathbb{F}, L_u, \mu, k]}(L_1[\mathbb{K}, \mathbb{F}, L_u, \mu, k])$ then $\dim(L_v) = \dim(L_1[\mathbb{K}, \mathbb{F}, L_u, \mu, k]) = \lceil \frac{\dim(L_u)}{2} \rceil$; (b) if there exists $\beta \in L_1[\mathbb{K}, \mathbb{F}, L_u, \mu, k]$ such that $L_v = L_\beta[\mathbb{K}, \mathbb{F}, L_u, \mu, k]$ then $\dim(L_v) = \dim(L_0[\mathbb{K}, \mathbb{F}, L_u, \mu, k]) + \mu = \lfloor \frac{\dim(L_u)}{2} \rfloor + \mu$. In either case $\dim(L_u) \cdot 2^{-1} \leq \dim(L_v) \leq \dim(L_u) \cdot 2^{-1} + \mu$, and the claim follows. □

Claim G.7. *The native cover $S[\mathbb{K}, \mathbb{F}, L, \mu, k]$ is $|\mathbb{K}|^{\frac{\dim(L)}{2} - \mu - 2}$-independent.*

Proof. Recalling Definition 4.16, fix arbitrary subsets $D' \subseteq (\{\mathrm{col}\} \times L'_0) \sqcup (\{\mathrm{row}\} \times L_1)$ and $D'' \subseteq D_\square[\mathbb{K}, \mathbb{F}, L, \mu]$ both of size at most $|\mathbb{K}|^{\frac{\dim(L)}{2} - \mu - 2}$, and define $\tilde{D} := D'' \cup (\cup_{(\mathrm{col},\alpha) \in D'} D_{\mathrm{col},\alpha}) \cup (\cup_{(\mathrm{row},\beta) \in D'} D_{\mathrm{row},\beta})$. Let $w' \in \mathbb{F}^{D^{\text{BS-RS}}}$ be such that: (i) for every $(\mathrm{col}, \alpha) \in D'$ it holds that $\psi_{\mathrm{col},\alpha}(w') \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$; and (ii) for every $(\mathrm{row}, \beta) \in D'$ it holds that $\psi_{\mathrm{row},\beta}(w') \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k]$. We need to show that there exists $w \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ such that $w|_{\tilde{D}} = w'|_{\tilde{D}}$.

In fact, it suffices to show that there exists $w_\square \in \mathrm{RS}_\square[\mathbb{K}, \mathbb{F}, L, \mu]$ such that $w_\square|_{\tilde{D} \cap D_\square} = w'|_{\tilde{D} \cap D_\square}$, because Claim F.4 implies there exists a unique codeword $w \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ such that $w|_{D_\square} = w_\square$ and $w|_{\tilde{D}} = w'|_{\tilde{D}}$.

Thus, we now argue that there exists $w_\square \in \mathrm{RS}_\square[\mathbb{K}, \mathbb{F}, L, \mu]$ such that the following holds.

- For every $(\mathrm{col}, \alpha) \in D'$ and $\beta \in Z_{L_0}(L_1)$, it holds that $f_{w_\square}(\alpha, \beta) = w'(\phi(\alpha, \beta)) = (\psi_{\mathrm{col},\alpha}(w'))(\mathrm{rs}, \beta)$. In particular, $f_{w_\square}|_{\{\alpha\} \times Z_{L_0}(L_1)} \in \mathrm{RS}[\mathbb{F}, Z_{L_0}(L_1), |L_1| \cdot \rho]$, and let $p_{\mathrm{col},\alpha}$ be its univariate low degree extension to $\mathbb{F}$.
- For every $(\mathrm{row}, \beta) \in D'$ and $\alpha \in L_\beta$, it holds that $f_{w_\square}(\alpha, Z_{L_0}(\beta)) = w'(\phi(\alpha, Z_{L_0}(\beta))) = (\psi_{\mathrm{row},\beta}(w'))(\mathrm{rs}, \alpha)$. In particular, $f_{w_\square}|_{L_\beta \times \{Z_{L_0}(L_1)\}} \in \mathrm{RS}[\mathbb{F}, L_\beta, |L_0|]$, and let $p_{\mathrm{row},\beta}$ be its univariate low degree extension to $\mathbb{F}$.
- For every $(\alpha, \beta) \in D''$, it holds that $f_{w_\square}(\alpha, Z_{L_0}(\beta)) = w'(\phi(\alpha, Z_{L_0}(\beta)))$.

By Definition F.2 it suffices to show that there exists a bivariate polynomial $g \in \mathbb{F}[X, Y]$ such that: (i) $\deg_X(g) < |L_0|$; (ii) $\deg_Y(g) < |L_1| \cdot \rho$; (iii) $g|_{X=\alpha} = p_{\mathrm{col},\alpha}$ for every $(\mathrm{col}, \alpha) \in D'$; (iv) $g|_{Y=Z_{L_0}(\beta)} = p_{\mathrm{row},\beta}$ for every $(\mathrm{row}, \beta) \in D'$; (v) $g(\alpha, \beta) = w'(\phi(\alpha, Z_{L_0}(\beta)))$ for every $(\alpha, \beta) \in D''$. But notice that $|D'| + |D''| \leq 2 \cdot |\mathbb{K}|^{\frac{\dim(L)}{2} - \mu - 2} = |\mathbb{K}|^{\frac{\dim(L)}{2} - \mu - 1} < \min\{|L_0|, |L_1| \cdot \rho\}$, because (a) $\log_{|\mathbb{K}|}(|L_0|) = \dim(L_0) \geq \frac{\dim(L)}{2} - 1$, and (b) $\log_{|\mathbb{K}|}(|L_1| \cdot \rho) = \dim(L_1) - \mu \geq \frac{\dim(L)}{2} - \mu$. The claim follows by considering a suitable interpolating set (see Section H). □

G.2 Computing spanning sets of dual codes in the recursive cover

We prove that spanning sets for duals of codes in the recursive cover can be computed efficiently; this is the key fact that we later use to argue that the algorithm required by Lemma 4.27 satisfies the stated time complexity.

Claim G.8. *For every positive integer m and vertex v in $T[\mathbb{K}, \mathbb{F}, L, \mu, k]$ of depth at least $\log_2 \dim(L) - \log_2 \log_{|\mathbb{K}|} m$, a spanning set of $\tilde{C}_v^{\perp}$ can be computed in time $\mathrm{poly}(\log_2 |\mathbb{F}| + |\mathbb{K}|^{\mu} + m)$.*

The above claim directly follows from Claim G.9 and Claim G.10, stated and proved below.

Claim G.9. *For every positive integer m and vertex v in $T[\mathbb{K}, \mathbb{F}, L, \mu, k]$ of depth at least $\log_2 \dim(L) - \log_2 \log_{|\mathbb{K}|} m$, $|\tilde{D}_v| \leq \mathrm{poly}(m + |\mathbb{K}|^{\mu})$.*

Proof. Ben-Sasson and Sudan [BS08] show that the block length of $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ is $\tilde{O}_{\mathbb{K},\mu,k}(|L|)$ for any fixed $\mathbb{K}, \mu, k$. One can verify that, if we do not fix these parameters, the block length is $\tilde{O}(|L| \cdot |\mathbb{K}|^{\mu})$. Next, observe that $\tilde{C}_v = \text{BS-RS}[\mathbb{K}, \mathbb{F}, L_v, \mu, k]$ for some L_v such that $\dim(L_v) \leq \log_{|\mathbb{K}|} m + 2\mu$ (Claim G.6); in this case, the aforementioned bound becomes $\mathrm{poly}(m + |\mathbb{K}|^{\mu})$. □

Claim G.10. *A spanning set for $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]^{\perp}$ can be found in time $\mathrm{poly}\left(\log_2 |\mathbb{F}| + |D^{\text{BS-RS}}[\mathbb{K}, \mathbb{F}, L, \mu, k]|\right)$.*

Proof. We show an algorithm that constructs the desired spanning set in the stated time complexity. First, a spanning set for $\mathrm{RS}[\mathbb{F}, S, d]^{\perp}$ can be found in time $\mathrm{poly}(\log_2 |\mathbb{F}| + |S|)$, for any finite field $\mathbb{F}$, subset $S \subseteq \mathbb{F}$, and degree bound $d < |S|$. Hence, a spanning set for $\mathrm{RS}_{\square}[\mathbb{K}, \mathbb{F}, L, \mu]^{\perp}$ can be found in time $(\log_2 |\mathbb{F}| \cdot |L| \cdot |\mathbb{K}|^{\mu})^c$ for some $c > 0$.

We argue by induction on $\dim(L)$ that a spanning set for $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]^{\perp}$ can be found in time $(\log_2 |\mathbb{F}| \cdot |L| \cdot |\mathbb{K}|^{\mu})^c$. We rely the property that the code $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ is covered by

$$\{(\text{BS-RS}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k], D_{\mathrm{col},\alpha})\}_{\alpha \in L_0'} \cup \{(\text{BS-RS}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k], D_{\mathrm{row},\beta})\}_{\beta \in L_1}$$

and the property that $w \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]$ if an only if:

- $\psi_{\mathrm{col},\alpha}(w) \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, Z_{L_0}(L_1), \mu, k]$ for every $\alpha \in L_0'$ and
- $\psi_{\mathrm{row},\beta}(w) \in \text{BS-RS}[\mathbb{K}, \mathbb{F}, L_\beta, \mu, k]$ for every $\beta \in L_1$.

Thus $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]^{\perp}$ is spanned by the duals of codes in its cover and, in particular, is spanned by their spanning sets; in sum, it suffices to construct a spanning set for its cover.

In light of the above, we can bound the construction time of a spanning set for $\text{BS-RS}[\mathbb{K}, \mathbb{F}, L, \mu, k]^{\perp}$ as follows:

- If $\dim(L) \leq k$, the claim follows as the code in this case simply equals $\mathrm{RS}_{\square}[\mathbb{K}, \mathbb{F}, L, \mu]$.
- If $\dim(L) > k$, the time to construct a spanning set is at most the time to construct spanning sets for the cover:

$$\begin{aligned}
&|L_0'| \cdot (\log_2 |\mathbb{F}| \cdot |L_1| \cdot |\mathbb{K}|^{\mu})^c + |L_1| \cdot (\log_2 |\mathbb{F}| \cdot |L_0'| \cdot |\mathbb{K}|^{\mu+1})^c && (1)\\
&= |L_0| \cdot |\mathbb{K}|^{\mu-1} \cdot (\log_2 |\mathbb{F}| \cdot |L_1| \cdot |\mathbb{K}|^{\mu})^c + |L_1| \cdot (\log_2 |\mathbb{F}| \cdot |L_0| \cdot |\mathbb{K}|^{2\mu})^c && (2)\\
&= |L| \cdot (\log_2 |\mathbb{F}| \cdot |\mathbb{K}|^{\mu})^c \cdot \left(|\mathbb{K}|^{\mu-1} \cdot |L_1|^{c-1} + |\mathbb{K}|^{c\mu} \cdot |L_0|^{c-1}\right) && (3)\\
&\leq |L| \cdot (\log_2 |\mathbb{F}| \cdot |\mathbb{K}|^{\mu})^c \cdot \left(|\mathbb{K}|^{\mu+c-2} \cdot |L|^{\frac{c-1}{2}} + |\mathbb{K}|^{c\mu} \cdot |L|^{\frac{c-1}{2}}\right) && (4)\\
&= |L|^{\frac{c+1}{2}} \cdot (\log_2 |\mathbb{F}| \cdot |\mathbb{K}|^{\mu})^c \cdot \left(|\mathbb{K}|^{\mu+c-2} + |\mathbb{K}|^{c\mu}\right) && (5)\\
&\leq (\log_2 |\mathbb{F}| \cdot |L| \cdot |\mathbb{K}|^{\mu})^c \ . && (6)
\end{aligned}$$

Above, (1) is by the inductive assumption, (2) is by definition of L_0', (3) is by the fact that $|L_0| \cdot |L_1| = |L|$, and (4) is by definition of L_0, L_1. We are left to show (6), and this follows from the fact that: (i) $\dim(L) > k$, (ii) $k > 2\mu$ by definition, and (iii) we can choose c to be large enough (namely, so that $|\mathbb{K}|^{\mu+c-2} + |\mathbb{K}|^{c\mu} \leq |L|^{\frac{c-1}{2}}$ holds). □

G.3 Putting things together

Proof of Lemma 4.27. Define the depth function $d(\mathbb{K}, L, \mu, a) := \log_2 \dim(L) - \log_2(\log_{|\mathbb{K}|} a + \mu + 2) - 1$. We argue the two conditions in the lemma. First, for every index $\mathtt{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$, $T[\mathbb{K}, \mathbb{F}, L, \mu, k]$ is a 1-intersecting recursive cover of BS-RS$[\mathbb{K}, \mathbb{F}, L, \mu, k]$ (by Claim G.4). Moreover, for every positive integer m and non-leaf vertex v in T with $\mathrm{depth}(T, v) < d(\mathbb{K}, L, \mu, m)$, the cover T_v is m-independent (by Claim G.5).

Second, consider the algorithm that, given an index $\mathtt{n} = (\mathbb{K}, \mathbb{F}, L, \mu, k)$ and subset $I \subseteq D^{\text{BS-RS}}[\mathbb{K}, \mathbb{F}, L, \mu, k]$, works as follows: (1) for every $\alpha \in I$ choose an arbitrary vertex v_α in $\mathrm{layer}(T, d(\mathbb{K}, L, \mu, |I|))$ such that $\alpha \in \tilde{D}_{v_\alpha}$, and then set $U := \{v_\alpha\}_{\alpha \in I}$; (2) compute a spanning set W_v set for $\tilde{C}_v^\perp$; (3) return $W := \cup_{u \in U} W_u$. This algorithm satisfies the required properties. First, it runs in time $\mathrm{poly}(\log_2 |\mathbb{F}| + \dim(L) + |\mathbb{K}|^\mu + |I|)$ because a spanning set set for $\tilde{C}_u^\perp$ can be computed in time $\mathrm{poly}(\log_2 |\mathbb{F}| + |\mathbb{K}|^\mu + |I|)$ (by Claim G.8). Next, its output W meets the requirements:

- $U \subseteq \mathrm{layer}(T[\mathbb{K}, \mathbb{F}, L, \mu, k], d(\mathbb{K}, L, \mu, |I|))$;
- $|U| \leq |I|$, by definition of U;
- $I \subseteq (\cup_{u \in U} \tilde{D}_u)$, by definition of U;
- $\mathrm{span}(W) = \mathrm{span}(\cup_{u \in U} W_u) = \mathrm{span}(\cup_{u \in U} \tilde{C}_u^\perp)$, by definition of W and W_u.

This completes the proof of Lemma 4.27. □

H Folklore claim on interpolating sets

Claim H.1. *Let $\mathbb{F}$ be a field, let $d_{\mathsf{cols}}, d_{\mathsf{rows}} \in \mathbb{N}$, and consider three sets $S_{\mathsf{cols}}, S_{\mathsf{rows}} \subseteq \mathbb{F}$ and $S_{\mathsf{pnts}} \subseteq \mathbb{F} \times \mathbb{F}$ such that $|S_{\mathsf{cols}}| + |S_{\mathsf{rows}}| + |S_{\mathsf{pnts}}| \leq \min\{d_{\mathsf{cols}}, d_{\mathsf{rows}}\}$. Let $f \colon \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ be a function such that:*

- *for every $\alpha \in S_{\mathsf{cols}}$ there exists $g_{\mathsf{col},\alpha} \in \mathbb{F}^{<d_{\mathsf{rows}}}[x]$ such that $f(\alpha, \beta) = g_{\mathsf{col},\alpha}(\beta)$ for every $\beta \in \mathbb{F}$;*
- *for every $\beta \in S_{\mathsf{rows}}$ there exists $g_{\mathsf{row},\beta} \in \mathbb{F}^{<d_{\mathsf{cols}}}[x]$ such that $f(\alpha, \beta) = g_{\mathsf{row},\beta}(\alpha)$ for every $\alpha \in \mathbb{F}$.*

Then there exists $g \in \mathbb{F}[X, Y]$ such that: (i) $\deg_X(g) < d_{\mathsf{rows}}$; (ii) $\deg_Y(g) < d_{\mathsf{cols}}$; (iii) $g|_{X=\alpha} = g_{\mathsf{col},\alpha}$ for every $\alpha \in S_{\mathsf{cols}}$; (iv) $g|_{Y=\beta} = g_{\mathsf{row},\beta}$ for every $\beta \in S_{\mathsf{rows}}$; (v) $g(\alpha, \beta) = f(\alpha, \beta)$ for every $(\alpha, \beta) \in S_{\mathsf{pnts}}$.

Proof. Any rectangle $D_X \times D_Y \subseteq \mathbb{F} \times \mathbb{F}$ with $|D_X| = d_{\mathsf{rows}}$ and $|D_Y| = d_{\mathsf{cols}}$ is an interpolating set: for every $w \in \mathbb{F}^{D_X \times D_Y}$ there exists a unique $g \in \mathbb{F}[X, Y]$ such that: (i) $\deg_X(g) < d_{\mathsf{rows}}$; (ii) $\deg_Y(g) < d_{\mathsf{cols}}$; (iii) $g|_{D_X \times D_Y} = w$. Define

$$D_X := S_{\mathsf{cols}} \cup \{\alpha : \exists\, \beta \text{ s.t. } (\alpha, \beta) \in S_{\mathsf{pnts}}\} \quad \text{and} \quad D_Y := S_{\mathsf{rows}} \cup \{\beta : \exists\, \alpha \text{ s.t. } (\alpha, \beta) \in S_{\mathsf{pnts}}\} \ .$$

Note that $|D_X| \leq d_{\mathsf{rows}}$ and $|D_Y| \leq d_{\mathsf{cols}}$; if either is strictly smaller, extend it arbitrarily to match the upper bound.

Choose $w \in \mathbb{F}^{D_X \times D_Y}$ to be a word that satisfies: (i) $w(\alpha, \beta) = f(\alpha, \beta)$ for every $\alpha \in S_{\mathsf{cols}}$ and $\beta \in D_Y$; (ii) $w(\alpha, \beta) = f(\alpha, \beta)$ for every $\beta \in S_{\mathsf{rows}}$ and $\alpha \in D_X$; (iii) $w(\alpha, \beta) = f(\alpha, \beta)$ for every $(\alpha, \beta) \in S_{\mathsf{pnts}}$. Denote by $g_w \in \mathbb{F}[X, Y]$ the unique "low degree extension" of w; we show that g_w satisfies the requirements of the claim.

The degree bounds and the equivalence on S_{pnts} follows by definition of g_w; thus it suffices to show equivalence of g_w with f when restricted to the required rows and columns.

- For every $\alpha \in S_{\mathsf{cols}}$: it holds by definition of g_w that $g_w|_{\{\alpha\} \times D_Y} = f|_{\{\alpha\} \times D_Y}$; moreover, $g_w|_{\{\alpha\} \times \mathbb{F}}$ and $f|_{\{\alpha\} \times \mathbb{F}}$ are evaluations of polynomials of degree less than $|D_Y|$, which implies that $g_w|_{\{\alpha\} \times \mathbb{F}} = f|_{\{\alpha\} \times \mathbb{F}}$.
- For every $\beta \in S_{\mathsf{rows}}$: it holds by definition of g_w that $g_w|_{D_X \times \{\beta\}} = f|_{D_X \times \{\beta\}}$; moreover, $g_w|_{\mathbb{F} \times \{\beta\}}$ and $f|_{\mathbb{F} \times \{\beta\}}$ are evaluations of polynomials of degree less than $|D_X|$, which implies that $g_w|_{\mathbb{F} \times \{\beta\}} = f|_{\mathbb{F} \times \{\beta\}}$. □

Acknowledgements

Work of E. Ben-Sasson, A. Gabizon, and M. Riabzev was supported by the Israel Science Foundation (grant 1501/14). Work of A. Chiesa and N. Spooner was partially supported in part by the UC Berkeley Center for Long-Term Cybersecurity. Work of M. A. Forbes was supported by the NSF, including NSF CCF-1617580, and the DARPA Safeware program; it was also partially completed when the author was at Princeton University, supported by the Princeton Center for Theoretical Computer Science.

References

[AH91] William Aiello and Johan Håstad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42(3):327–345, 1991. Preliminary version appeared in FOCS '87.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.

[AR16] Benny Applebaum and Pavel Raykov. On the relationship between statistical zero-knowledge and statistical randomized encodings. In *Proceedings of the 36th Annual International Cryptology Conference*, CRYPTO '16, pages 449–477, 2016.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003. Preliminary version appeared in STOC '97.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, 1985.

[BCG+17] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Interactive oracle proofs with constant rate and query complexity. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming*, ICALP '17, pages 40:1–40:15, 2017.

[BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *Proceedings of the 13th Theory of Cryptography Conference*, TCC '16-A, pages 33–64, 2016.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Proceedings of the 14th Theory of Cryptography Conference*, TCC '16-B, pages 31–60, 2016.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. Preliminary version appeared in FOCS '90.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Proceedings of the 8th Annual International Cryptology Conference*, CRYPTO '89, pages 37–56, 1988.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[BGK+10] Eli Ben-Sasson, Venkatesan Guruswami, Tali Kaufman, Madhu Sudan, and Michael Viderman. Locally testable codes require redundant testers. *SIAM Journal on Computing*, 39(7):3230–3247, 2010.

[BGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 113–131, 1988.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, 1988.

[BHR05] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005.

[BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

[BVW98] Ronald V. Book, Heribert Vollmer, and Klaus W. Wagner. Probabilistic type-2 operators and "almost"-classes. *Computational Complexity*, 7(3):265–289, 1998.

[BW04] Andrej Bogdanov and Hoeteck Wee. A stateful implementation of a random function supporting parity queries over hypercubes. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, and of the 8th International Workshop on Randomization and Computation*, APPROX-RANDOM '04, pages 298–309, 2004.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017.

[DFK+92] Cynthia Dwork, Uriel Feige, Joe Kilian, Moni Naor, and Shmuel Safra. Low communication 2-prover zero-knowledge proofs for NP. In *Proceedings of the 11th Annual International Cryptology Conference*, CRYPTO '92, pages 215–227, 1992.

[DR04] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

[DS98] Cynthia Dwork and Amit Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In *Proceedings of the 18th Annual International Cryptology Conference*, CRYPTO '98, pages 442–457, 1998.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in FOCS '91.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge (extended abstract). In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 204–209, 1987.

[FRS88] Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. In *Theoretical Computer Science*, pages 156–161, 1988.

[FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In *Proceedings of the 9th Annual International Cryptology Conference*, CRYPTO '89, pages 526–544, 1989.

[GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7):2761–2822, 2010. Preliminary version appeared in FOCS '03.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 173–190, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for Muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 113–122, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version appeared in STOC '85.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991. Preliminary version appeared in FOCS '86.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 6th Innovations in Theoretical Computer Science Conference*, ITCS '15, pages 133–142, 2015.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM*, 53:558–655, July 2006. Preliminary version in STOC '02.

[GV99] Oded Goldreich and Salil P. Vadhan. Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, CCC '99, page 54, 1999.

[IMS12] Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. On efficient zero-knowledge PCPs. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 151–168, 2012.

[IMSX15] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. On zero-knowledge PCPs: Limitations, simplifications, and applications, 2015. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`.

[IOS97] Toshiya Itoh, Yuji Ohta, and Hiroki Shizuya. A language-dependent cryptographic primitive. *Journal of Cryptology*, 10(1):37–50, 1997.

[IW14] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *Proceedings of the 11th Theory of Cryptography Conference*, TCC '14, pages 121–145, 2014.

[IWY16] Yuval Ishai, Mor Weiss, and Guang Yang. Making the best of a leaky situation: Zero-knowledge PCPs from leakage-resilient circuits. In *Proceedings of the 13th Theory of Cryptography Conference*, TCC '16-A, pages 3–32, 2016.

[IY87] Russell Impagliazzo and Moti Yung. Direct minimum-knowledge computations. In *Proceedings of the 7th Annual International Cryptology Conference*, CRYPTO '87, pages 40–51, 1987.

[Kay10] Neeraj Kayal. Algorithms for arithmetic circuits, 2010. ECCC TR10-073.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KPT97] Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 496–505, 1997.

[KR08] Yael Kalai and Ran Raz. Interactive PCP. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, ICALP '08, pages 536–547, 2008.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[LS95] Dror Lapidot and Adi Shamir. A one-round, two-prover, zero-knowledge protocol for NP. *Combinatorica*, 15(2):204–214, 1995.

[MX13] Mohammad Mahmoody and David Xiao. Languages with efficient zero-knowledge PCPs are in SZK. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 297–314, 2013.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version appeared in CRYPTO '89.

[Nis93] Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

[Oka00] Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. *Journal of Computer and System Sciences*, 60(1):47–108, 2000.

[Ost91] Rafail Ostrovsky. One-way functions, hard on average problems, and statistical zero-knowledge proofs. In *Proceedings of the 6th Annual Structure in Complexity Theory Conference*, CoCo '91, pages 133–138, 1991.

[OV08] Shien Jin Ong and Salil P. Vadhan. An equivalence between zero knowledge and commitments. In *Proceedings of the 5th Theory of Cryptography Conference*, TCC '08, pages 482–500, 2008.

[OW93] Rafail Ostrovsky and Avi Wigderson. One-way functions are essential for non-trivial zero-knowledge. In *Proceedings of the 2nd Israel Symposium on Theory of Computing Systems*, ISTCS '93, pages 3–17, 1993.

[RRR16] Omer Reingold, Ron Rothblum, and Guy Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th ACM Symposium on the Theory of Computing*, STOC '16, pages 49–62, 2016.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005. Preliminary version appeared in CCC '04.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[SV03] Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, 2003.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[Vad99] Salil P. Vadhan. *A Study of Statistical Zero-Knowledge Proofs*. PhD thesis, MIT, August 1999.

[VV15] Vinod Vaikuntanathan and Prashant Nalini Vasudevan. Secret sharing and statistical zero knowledge. In *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '15, pages 656–680, 2015.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the 1979 International Symposium on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, 1979.