A Zero Knowledge Sumcheck and its Applications

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Michael A. Forbes
miforbes@csail.mit.edu
Simons Institute for the Theory of Computing

Nicholas Spooner
nick.spooner@berkeley.edu
University of Toronto and UC Berkeley

April 6, 2017

Abstract

Many seminal results in Interactive Proofs (IPs) use algebraic techniques based on low-degree polynomials, the study of which is pervasive in theoretical computer science. Unfortunately, known methods for endowing such proofs with zero knowledge guarantees do not retain this rich algebraic structure.

In this work, we develop algebraic techniques for obtaining zero knowledge variants of proof protocols in a way that leverages and preserves their algebraic structure. Our constructions achieve unconditional (perfect) zero knowledge in the Interactive Probabilistically Checkable Proof (IPCP) model of Kalai and Raz [KR08] (the prover first sends a PCP oracle, then the prover and verifier engage in an Interactive Proof in which the verifier may query the PCP).

Our main result is a zero knowledge variant of the sumcheck protocol [LFKN92] in the IPCP model. The sumcheck protocol is a key building block in many IPs, including the protocol for polynomial-space computation due to Shamir [Sha92], and the protocol for parallel computation due to Goldwasser, Kalai, and Rothblum [GKR15]. A core component of our result is an algebraic commitment scheme, whose hiding property is guaranteed by algebraic query complexity lower bounds [AW09; JKRS09]. This commitment scheme can then be used to considerably strengthen our previous work [BCFGRS16] that gives a sumcheck protocol with much weaker zero knowledge guarantees, itself using algebraic techniques based on algorithms for polynomial identity testing [RS05; BW04].

We demonstrate the applicability of our techniques by deriving zero knowledge variants of well-known protocols based on algebraic techniques. First, we construct zero knowledge IPCPs for $\mathbf{NEXP}$ starting with the Multi-prover Interactive Proofs of Babai, Fortnow, and Lund [BFL91]. This result is a direct application of our zero knowledge sumcheck and our algebraic commitment scheme, augmented with the use of 'randomized' low-degree extensions.

We also construct protocols in a more restricted model where the prover and verifier engage in a standard Interactive Proof with oracle access to a uniformly random low-degree polynomial (soundness holds with respect to *any* oracle). In this setting we achieve zero knowledge variants of the protocols of Shamir and of Goldwasser, Kalai, and Rothblum.

Keywords: zero knowledge; sumcheck; algebraic query complexity; probabilistically checkable and interactive proofs

Contents

1 Introduction

The notion of *Interactive Proofs* (IPs) [BM88; GMR89] is fundamental in Complexity Theory and Cryptography. An Interactive Proof for a language $\mathscr{L}$ is a protocol between a probabilistic polynomial-time *verifier* and a resource-unbounded *prover* that works as follows: given a common input $\mathtt{x}$, the prover and verifier exchange some number of messages and then the verifier either accepts or rejects. If $\mathtt{x}$ is in $\mathscr{L}$ then the verifier always accepts; if instead $\mathtt{x}$ is not in $\mathscr{L}$ then the verifier rejects with high probability, regardless of the prover's actions. The seminal results of Lund, Fortnow, Karloff, and Nisan [LFKN92] and Shamir [Sha92] demonstrate the surprising expressiveness of Interactive Proofs, in particular showing that every language decidable in polynomial space has an Interactive Proof.

Research on IPs has recently focused on new and more refined goals, motivated by the paradigm of *delegation of computation*, in which a resource-limited verifier receives the help of a resource-rich prover to check the output of an expensive (but tractable) computation. In this setting bounding the complexity of the honest prover is important. While every IP protocol has a polynomial-space prover, this prover may run in superpolynomial time, even if the protocol is for a tractable language. Recent work has focused on *doubly-efficient* IPs, where the prover is efficient (it runs in polynomial time) and the verifier is highly efficient (it runs in, say, quasilinear time). Doubly-efficient IPs can be achieved, with various tradeoffs, for many types of computation: languages decidable by uniform circuits of polylogarithmic depth [GKR15]; languages decidable in polynomial time and bounded-polynomial space [RRR16]; and languages decidable by conjunctions of polynomially-many 'local' conditions [RG17].

A key building block in *all* of these protocols is the *sumcheck protocol* [LFKN92], which is an Interactive Proof for claims of the form "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$", where H is a subset of a finite field $\mathbb{F}$ and F is an m-variate polynomial over $\mathbb{F}$ of small individual degree. The use of sumcheck imbues the aforementioned protocols with an algebraic structure, where the verifier *arithmetizes* a boolean problem into a statement about low-degree polynomials, which can then be checked via the sumcheck protocol. This algebraic structure is not only elegant, but also very useful. Indeed, this structure is crucial not only for highly-efficient software and hardware systems for delegating computation [CMT12; TRMP12; Tha13; Tha15; WHGSW16; WJBSTWW17] but also for a diverse set of compelling theoretical applications such as memory delegation [CKLR11], refereed delegation [CRR13], IPs of proximity [RVW13], and many others.

Despite these subsequent works demonstrating the flexibility of sumcheck to accommodate additional desirable properties, the *zero knowledge* [GMR89] properties of sumcheck have not been explored. This is surprising because zero knowledge, the ability of the prover to establish the validity of an assertion while revealing no insight into its proof, is highly desirable for the cryptographic applications of Interactive Proofs. Unfortunately, achieving zero knowledge is nontrivial because the sumcheck protocol reveals the results of intermediate computations, in particular the partial sums $\sum_{\vec{\alpha} \in H^{m-i}} F(c_1, \ldots, c_i, \vec{\alpha})$ for $c_1, \ldots, c_i \in \mathbb{F}$ chosen by the verifier. These partial sums are in general $\#\mathbf{P}$-hard to compute so they convey significant additional knowledge to the verifier.

The goal of this work is to enlarge the existing algebraic toolkit based on low-degree polynomials and use these tools to provide a native extension of the sumcheck protocol that is zero knowledge, and to explore applications of such an extension. As we discuss shortly, however, we cannot expect to do so within the model of Interactive Proofs.

1.1 Prior techniques for achieving zero knowledge

We briefly describe why existing methods fall short of our goal, which is making the sumcheck protocol zero knowledge in an algebraic way. A seminal result in cryptography says that if one-way functions exist then every language having an IP also has a *computational* zero knowledge IP [GMR89; IY87; BGGHKMR88]; this assumption is 'minimal' in the sense that if one-way functions do not exist then computational zero knowledge IPs capture only "average-case" $\mathbf{BPP}$ [Ost91; OW93]. While powerful, such results are unsatisfactory from our perspective. First, cryptography adds significant efficiency overheads, especially when used in a non-blackbox way as these results do. Second, the results rely on transformations that erase all the algebraic structure of the underlying protocols. While these limitations can be mitigated by using cryptography that leverages some of the underlying structure [CD98], the costs incurred by the use of cryptography remain significant. Ideally, we wish to *avoid* intractability assumptions.

Unfortunately, this is impossible to achieve under standard complexity assumptions, because Interactive Proofs that are *statistical* zero knowledge are limited to languages in $\mathbf{AM} \cap \mathbf{coAM}$ [For87; AH91]. Such languages (conjecturally) do not even include $\mathbf{NP}$, so that we cannot even hope to achieve a 'zero knowledge sumcheck protocol' (which would give $\#\mathbf{P}$).

The quest for zero knowledge without relying on intractability assumptions led to the formulation of *Multi-prover Interactive Proofs* (MIPs) [BGKW88], where the verifier exchanges messages with two or more non-communicating provers. Groundbreaking results establish that MIPs are very powerful: all (and only) languages decidable in non-deterministic exponential time have MIPs [BFL91] and, in fact, even *perfect* zero knowledge MIPs [BGGHKMR88; DFKNS92]. Similar results hold even for the related model of *Probabilistically Checkable Proofs* (PCPs) [FRS88; BFLS91; FGLSS96; AS98; ALMSS98], where the prover outputs a proof string that the verifier can check by reading only a few randomly-chosen locations. Namely, all (and only) languages decidable in non-deterministic exponential time have PCPs [BFLS91] and, in fact, even *statistical* zero knowledge PCPs [KPT97; IMSX15].

However, while information-theoretic, the aforementioned works rely on transformations that, once again, discard the rich algebraic structure of the underlying protocols. Thus, zero knowledge in this setting continues to be out of reach of simple and elegant algebraic techniques.

1.2 Our goal: algebraic techniques for zero knowledge

Our goal is to develop information-theoretic techniques for achieving zero knowledge in a way that leverages, and preserves, the algebraic structure of the sumcheck protocol and other protocols that build on it. An additional goal is to preserve the simplicity and elegance of these foundational protocols.

Yet, as discussed, we cannot hope to do so with Interactive Proofs, and so we work in another model. We choose to work in a model that combines features of both Interactive Proofs and PCPs: the *Interactive PCP* (IPCP) model of Kalai and Raz [KR08]. The prover first sends to the verifier a long string as a PCP oracle, after which the prover and verifier engage in an Interactive Proof. The verifier is free at any point to query the PCP oracle at locations of its choice, and the verifier only pays for the number of queries it makes, so that exponentially-large PCP oracles are allowed.

Kalai and Raz [KR08] show that the IPCP model has efficiency advantages over both PCPs and IPs (individually). Goyal, Ishai, Mahmoody, and Sahai [GIMS10] construct efficient zero knowledge IPCPs, but their techniques mirror those for zero knowledge PCPs and, in particular, are not algebraic.

One can think of the IPCP model as lying somewhere 'in between' the IP and MIP models. Indeed, it is equivalent to a (2-prover) MIP where one of the provers is stateless (its answers do not depend on the verifier's prior messages or queries). This means that soundness is easier to achieve for an IPCP than for an MIP. Zero knowledge, however, is more difficult for an IPCP than for an MIP, because the stateless prover cannot choose which queries it will answer.

A significant advantage of the IPCP model over the MIP model is that one can easily compile (public-coin) IPCPs into cryptographic proofs via transformations that preserve zero knowledge, while only making a black-box use of cryptography. For example, using collision-resistant functions one can obtain public-coin interactive arguments by extending ideas of [Kil92; IMSX15]; also, using random oracles one can obtain publicly-verifiable non-interactive arguments via [BCS16] (extending the Fiat–Shamir paradigm [FS86] and Micali's "CS proofs" [Mic00]). In contrast, known transformations for MIPs yield private-coin arguments [BC12], or do not preserve zero knowledge [KRR14].

1.3 Main result: a zero knowledge sumcheck

Our main result is a zero knowledge analogue of the sumcheck protocol [LFKN92], a key building block in many protocols. We now informally state and discuss this result, and in the next sub-section we discuss its applications.

The goal of the sumcheck protocol is to efficiently verify claims of the form "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$", where H is a subset of a finite field $\mathbb{F}$ and F is an m-variate polynomial over $\mathbb{F}$ of low individual degree. As the sumcheck protocol is often used in situations where the polynomial F is only implicitly defined (including this paper), it is helpful to adopt the viewpoint of Meir [Mei13], regarding the sumcheck protocol as a *reduction* from the summation "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$" to an evaluation "$F(\vec{c}) = b$"; the latter can be checked directly by the verifier or by another protocol. The verifier in this reduction does not need any information about F, aside from knowing that F has small individual degree. The completeness property of the reduction is that if the summation claim is true, then so is the evaluation claim with probability one. Its soundness property is that if the summation claim is false, then so is the evaluation claim with high probability.

The theorem below states the existence of sumcheck protocol in the above sense that works in the IPCP model and is *zero knowledge*, which means that a verifier does not learn any information beyond the fact that F sums to 0 on H^m (and, in our case, a single evaluation of F). As usual, this means that we establish an efficient procedure for simulating

the interaction of a (possibly malicious) verifier with the prover, where the simulator only uses the knowledge of the sum of F on H^m (and a single evaluation of F) but otherwise has no actual access to the prover (or F). This interaction is a random variable depending on the randomness of both the verifier and the prover, and we show that the simulated interaction perfectly replicates this random variable.

Theorem 1.1 (Informal version of Theorem 6.4)**.** *There exists an IPCP for sumcheck with the following zero knowledge guarantee: the view of any probabilistic polynomial-time verifier in the protocol can be perfectly and efficiently simulated by a simulator that makes only a single query to F. Moreover, we do not require the full power of the IPCP model: the honest prover's PCP consists only of a random multi-variate polynomial over $\mathbb{F}$ of small individual degree.*

Our result significantly strengthens the IPCP for sumcheck of [BCFGRS16] (co-authored by this paper's authors), which is only zero knowledge with respect to a simulator which requires *unrestricted* query access to F. Namely, in order to simulate a verifier's view, their simulator must make a number of queries to F that equals the number of queries to the PCP oracle made by the verifier. This zero knowledge guarantee is weaker than the above because a malicious verifier can make an *arbitrarily-large* (but polynomial) number of queries to the PCP oracle. However, this weaker guarantee suffices in some cases, such as in the previous work [BCFGRS16].

Perhaps more damaging is that when using this 'weakly zero knowledge' sumcheck protocol recursively (as required in applications), we would incur an *exponential blowup*: each simulated query recursively requires further queries to be simulated. In contrast, the 'strongly zero knowledge' sumcheck protocol that we achieve only requires the simulator to make a single query to F regardless of the malicious verifier's runtime, both providing a stronger zero knowledge guarantee and avoiding any exponential blow-up.

An important property of our sumcheck protocol (which also holds for [BCFGRS16]), is that it suffices for the honest prover to send as an oracle a uniformly random polynomial of a certain arity and degree. This brings the result 'closer to IP', in the sense that while IPCP captures all of $\mathbf{NEXP}$, only languages in $\mathbf{PSPACE}$ have IPCPs (with perfect completeness) where the honest prover behaves in this way. The same property holds for some of our applications.

Algebraic commitments. As detailed in Section 2, a key ingredient of our result is a commitment scheme based on algebraic techniques. That is, the prover wishes to commit to a value $b \in \mathbb{F}$, and to do so sends a PCP oracle to the verifier. To then reveal (decommit) b, the prover and verifier engage in an Interactive Proof. We show that the sumcheck protocol naturally yields such a commitment scheme, where the PCP oracle is simply a random low-degree polynomial R such that $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha}) = b$. The soundness guarantee of the sumcheck protocol shows that this commitment scheme is binding, so that the prover cannot "reveal" a value other than the correct b. To establish the hiding property, which states that the verifier cannot learn anything about b before the prover reveals it, we leverage lower bounds on the algebraic query complexity of polynomial summation, previously studied for completely different reasons [AW09; JKRS09].

As our commitments are themselves defined by low degree polynomials, they are 'transparent' to low degree testing. That is, in various protocols the prover sends to the verifier the evaluation table of a low-degree polynomial as a PCP oracle, and the verifier ensures that this evaluation table is (close to) low degree via low degree testing. In our zero knowledge setting, we need the prover to hide the evaluation table under a commitment (to be revealed selectively), and yet still allow the verifier to check that the underlying evaluation table represents a low-degree polynomial. Our commitments naturally have this property due to their algebraic structure, which we exploit in our applications discussed below.

Overall, the methods of this paper not only significantly deviate from traditional methods for achieving zero knowledge but also further illustrate the close connection between zero knowledge and Algebraic Complexity Theory, the theory of efficient manipulations of algebraic circuits and low-degree polynomials. This connection was first seen in our prior work developing the 'weakly zero knowledge sumcheck' of [BCFGRS16], used here as a subroutine. Indeed, this subroutine derives its zero-knowledge guarantee from an efficient algorithm for adaptively simulating random low-degree polynomials [BCFGRS16; BW04]. This algorithm itself relies on deterministic algorithms for polynomial identity testing of certain restricted classes of algebraic circuits [RS05]. We believe that it is an exciting research direction to further investigate this surprising connection, and to further broaden the set of information-theoretic algebraic techniques that are useful towards zero knowledge.

1.4 Applications: delegating computation in zero knowledge

The original sumcheck protocol (without zero knowledge) has many applications, including to various methods of delegating computation. Our zero knowledge sumcheck protocol can be used to obtain zero knowledge analogues of foundational results in this area: we achieve natural zero knowledge extensions of the first construction of PCPs/MIPs [BFL91; BFLS91], Shamir's protocol [Sha92], and doubly-efficient Interactive Proofs for low-depth circuits [GKR15].

1.4.1 Delegating non-deterministic exponential time

One of the earliest and most influential applications of the sumcheck protocol is the construction of Multi-prover Interactive Proofs for $\mathbf{NEXP}$ due to Babai, Fortnow, and Lund [BFL91]; the same construction also demonstrated the power of low-degree testing as a tool for checking arbitrary computations, another highly influential insight. The subsequent improvements by Babai, Fortnow, Levin, and Szegedy [BFLS91] led to the formulation and the study of *Probabilistically-Checkable Proofs* [BFLS91; FGLSS96] and then the celebrated PCP Theorem [AS98; ALMSS98].

We show how, by using our zero knowledge sumcheck protocol, we can obtain a zero knowledge analogue of the classical constructions of [BFL91; BFLS91].

Theorem 1.2 (Informal version of Theorem 7.2)**.** $\mathbf{NEXP}$ *has perfect zero knowledge Interactive PCPs.*

Our construction extends the protocol of [BFL91; BFLS91], which can be viewed as an IPCP that is later 'compiled' into an MIP or a PCP. This protocol reduces the $\mathbf{NEXP}$-complete problem of determining the satisfiability of a 'succinct' 3CNF, to testing whether there exists a low-degree polynomial satisfying an (exponentially large) set of constraints. A polynomial satisfying these constraints is necessarily a low-degree extension of a satisfying assignment, and thus implies the existence of such an assignment. The prover sends such a polynomial as an oracle, which the verifier then low-degree tests. That the constraints are satisfied can be checked using the sumcheck protocol.

To make this protocol zero knowledge we need to ensure that the oracle hides the original witness. We achieve this by sending not the low-degree extension itself but an algebraic commitment to it. The zero knowledge sumcheck then reduces the problem of checking the constraint on this witness to a single evaluation point of the low degree extension. However, this itself is not zero knowledge as evaluations of the low-degree extension can reveal information about the witness, especially if this evaluation is over the interpolating set H^m. Thus, our construction exploits the fact that the sumcheck protocol works for *any* low-degree extension of the witness, and not just the one of minimal degree. Thus, the prover will instead send (the commitment to) a randomly sampled extension of the witness of slightly higher (but still constant) individual degree. The evaluations of this polynomial will (outside the interpolating set H^m) be $O(1)$-wise independent. As the sumcheck reduction will reduce to an evaluation point outside H^m with high probability, the prover can then decommit the evaluation at this point to complete the sumcheck protocol without revealing any non-trivial information. We discuss this construction in more detail in Section 2.1.

1.4.2 Delegating polynomial space

The above result shows that a powerful prover can convince a probabilistic polynomial-time verifier of $\mathbf{NEXP}$ statements in perfect zero knowledge in the IPCP model. Now we turn our attention to protocols where the honest prover need not be a $\mathbf{NEXP}$ machine. One such protocol, due to Shamir [Sha92], provides an Interactive Proof for the $\mathbf{PSPACE}$-complete True Quantified Boolean Formula (TQBF) problem. This protocol is a more sophisticated application of the sumcheck protocol because sumcheck is applied *recursively*.

We aim to obtain zero knowledge analogues for these types of more complex protocols as well but now, to tackle the greater complexity, we proceed in two steps. First, we design a generic framework called *sum-product circuits* (which we believe to be of independent interest) that can express in a unified way a large class of 'sumcheck-based Interactive Proofs', such as Shamir's protocol. Second, we show how to use our zero knowledge sumcheck protocol to obtain zero knowledge analogues of these, and thus also for Shamir's protocol. We discuss this further in Section 2.5.

As before, the resulting protocols are within the IPCP model. However, a key feature of these protocols that differentiates them from our result for $\mathbf{NEXP}$ is that the prover *does not need the full power of IPCPs*: it suffices for the honest prover to send a PCP oracle that is the evaluation table of a random low-degree polynomial. Of course, soundness will continue to hold against any malicious prover that uses the PCP oracle in arbitrary ways.

Theorem 1.3 (Informal version of Theorem 10.2)**.** $\mathbf{PSPACE}$ *has perfect zero knowledge Interactive PCPs, where the honest prover sends a random low-degree polynomial as the oracle.*

As discussed above, any language having an IPCP where the honest prover sends a random low-degree polynomial (and achieves perfect completeness) can be decided in $\mathbf{PSPACE}$. In contrast, in the general case, deciding languages having IPCPs is $\mathbf{NEXP}$-hard. This result shows that moreover, all that is required to achieve unconditional zero knowledge for $\mathbf{PSPACE}$ is the ability to send a uniformly random polynomial as an oracle.

1.4.3 Delegating low-depth circuits

The doubly-efficient Interactive Proofs for low-depth circuits due to Goldwasser, Kalai, and Rothblum [GKR15] are another landmark result that makes a recursive use of the sumcheck protocol. Their construction can be viewed as a 'scaled down' version of Shamir's protocol where the prover is efficient and the verifier is highly efficient.

We obtain a zero knowledge analogue of this protocol; again it suffices for the honest prover to send a random low-degree polynomial as the oracle (and soundness holds against any oracle). We do so by showing how the computation of low-depth circuits can be reduced to a corresponding sum-product circuit, by following the arithmetization in [GKR15]; then we rely on our zero knowledge results for sum-product circuits, mentioned above.

The protocol of [GKR15] is an IP for delegating certain *tractable* computations: the evaluation of log-space uniform $\mathbf{NC}$ (circuits of polynomial size and polylogarithmic depth). The prover runs in polynomial time, while the verifier runs in quasilinear time and logarithmic space. But what does achieving zero knowledge mean in this case? If the simulator can run in polynomial time, then it can trivially simulate the verifier's view by simply running the honest prover. We thus need to consider a fine-grained notion of zero knowledge, by analyzing in more detail the overhead incurred by the simulator with respect to the malicious verifier's running time. This reckoning is similar to [BRV17], who study zero knowledge for Interactive Proofs of Proximity, and is a relaxation of *knowledge tightness* [Gol01, Section 4.4.4.2].

Concretely, we show that the running time of our simulator is a fixed (and small) polynomial in the verifier's running time, with only a polylogarithmic dependence on the size of the circuit. For example, the view of a malicious verifier running in time, say, $O(n^2)$ can be simulated in time $\tilde{O}(n^6)$. If the circuit has size $O(n^8)$, then the zero knowledge guarantee is meaningful because the simulator is not able to evaluate the circuit.

Theorem 1.4 (Informal version of Theorem 11.1)**.** *Log-space uniform* $\mathbf{NC}$ *has perfect zero knowledge Interactive PCPs, where the honest prover sends a random low-degree polynomial as the oracle, and the verifier runs in quasilinear time and logarithmic space. The simulator overhead is a small polynomial: the view of a verifier running in time* T *can be simulated in time* $T^3 \cdot \mathrm{polylog}(n)$.

An interesting open problem is whether the simulator overhead can be reduced to $T \cdot \mathrm{polylog}(n)$, as required in the (quite strict) definition given in [Gol01, Section 4.4.4.2]. It seems that our techniques are unlikely to achieve this because they depend on solving systems of linear equations in $\Omega(T)$ variables.

Finally, a property in [GKR15] that has been very useful in subsequent work is that the verifier only needs to query a single point in the low-degree extension of its input. In this case, the verifier runs in polylogarithmic time and logarithmic space. Our zero knowledge analogue retains these properties. Additionally, in this setting the size of the circuit is subexponential in the running time of the verifier. Our zero knowledge guarantee then implies that we obtain zero knowledge under the standard (not fine-grained) definition.

2 Techniques

We summarize the techniques underlying our contributions. We begin in Section 2.1 by recalling the protocol of Babai, Fortnow, and Lund, in order to explain its sources of information leakage and how one could prevent them via algebraic techniques. This discussion motivates the goal of an *algebraic* commitment scheme, described in Section 2.2. Then in Section 2.3 we explain how to use this tool to obtain our main result, a zero knowledge sumcheck protocol.

The rest of the section is then dedicated to explaining how to achieve our other applications, which involve achieving zero knowledge for *recursive* uses of the sumcheck protocol. First we explain in Section 2.4 what are the challenges that arise with regard to zero knowledge in recursive invocations of the sumcheck protocol, such as in the protocol of Shamir. Then in Section 2.5 we describe the framework of sum-product circuits, and the techniques within it that allows us to achieve zero knowledge for the protocols of Shamir and of Goldwasser, Kalai, and Rothblum.

2.1 An algebraic approach for zero knowledge in the BFL protocol

We recall the protocol of Babai, Fortnow, and Lund [BFL91] ('BFL protocol'), in order to explain its sources of information leakage and how one could prevent them via algebraic techniques. These are the ideas that underlie our algebraic construction of an unconditional (perfect) zero knowledge IPCP for $\mathbf{NEXP}$ (see Section 1.4.1).

The BFL protocol, and why it leaks. The O3SAT problem is the following $\mathbf{NEXP}$-complete problem: given a boolean formula B, does there exist a boolean function A such that

$$B(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0 \quad \text{for all } z \in \{0,1\}^r, b_1, b_2, b_3 \in \{0,1\}^s \ ?$$

The BFL protocol constructs an IPCP for O3SAT and later converts it to an MIP. Only the first step is relevant for us.

In the BFL protocol, the honest prover first sends a PCP oracle $\hat{A}\colon \mathbb{F}^s \to \mathbb{F}$ that is the unique multilinear extension (in some finite field $\mathbb{F}$) of a valid witness $A\colon \{0,1\}^s \to \{0,1\}$. The verifier must check that (a) $\hat{A}$ is a boolean function on $\{0,1\}^s$, and (b) $\hat{A}$'s restriction to $\{0,1\}^s$ is a valid witness for B. To do these checks, the verifier arithmetizes B into an arithmetic circuit $\hat{B}$, and reduces the checks to conditions that involve $\hat{A}$, $\hat{B}$, and other low-degree polynomials. A technique of [BFLS91] allows the verifier to 'bundle' all of these conditions into a low-degree polynomial f such that (with high probability over the choice of f) the conditions hold if and only if f sums to 0 on $\{0,1\}^{r+3s+3}$. The verifier checks that this is the case via a sumcheck protocol with the prover. The soundness of the sumcheck protocol depends on the PCP oracle being the evaluation of a low-degree polynomial; the verifier checks this using a low-degree test.

We see that the BFL protocol is *not* zero knowledge for two reasons: (i) the verifier has oracle access to $\hat{A}$ and, in particular, to the witness A; (ii) the prover's messages during the sumcheck protocol leak further information about A (namely, hard-to-compute partial sums of f, which itself depends on A).

A blueprint for zero knowledge. We now describe the 'blueprint' for an approach to achieve zero knowledge in the BFL protocol. The prover does not send $\hat{A}$ directly but instead a *commitment* to it. After this, the prover and verifier engage in a sumcheck protocol with suitable zero knowledge guarantees; at the end of this protocol, the verifier needs to evaluate f at a point of its choice, which involves evaluating $\hat{A}$ at three points. Now the prover reveals the requested values of $\hat{A}$, without leaking any information beyond these, so that the verifier can perform its check. We explain how these ideas motivate the need for certain algebraic tools, which we later obtain and use to instantiate our approach.

(1) Randomized low-degree extension. Even if the prover reveals only three values of $\hat{A}$, these may still leak information about A. We address this problem via a *randomized low-degree extension*. Indeed, while the prover in the BFL protocol sends the *unique* multilinear extension of A, one can verify that *any* extension of A of sufficiently low degree also works. We exploit this flexibility as follows: the prover randomly samples $\hat{A}$ in such a way that any three evaluations of $\hat{A}$ do not reveal any information about A. Of course, if any of these evaluations is within $\{0,1\}^s$, then no extension of A has this property. Nevertheless, during the sumcheck protocol, the prover can ensure that the verifier chooses only evaluations outside of $\{0,1\}^s$ (by aborting if the verifier deviates), which incurs only a small increase in the soundness error. With this modification in place, it suffices for the prover to let $\hat{A}$ be a random degree-4 extension of A: by a dimensionality argument, any 3 evaluations outside of $\{0,1\}^s$ are now independent and uniformly random in $\mathbb{F}$. Remarkably, we are thus able to reduce a claim about A to a claim which contains *no information* about A.

(2) Low-degree testing the commitment. The soundness of the sumcheck protocol relies on f having low degree, or at least being close to a low-degree polynomial. This in turn depends on the PCP oracle $\hat{A}$ being close to a low-degree

polynomial. If the prover sends $\hat{A}$, the verifier can simply low-degree test it. However, if the prover sends a commitment to $\hat{A}$, then it is not clear what the verifier should do. One option would be for the prover to reveal *more* values of $\hat{A}$ (in addition to the aforementioned three values), in order to enable the verifier to conduct its low-degree test on $\hat{A}$. The prover would then have to ensure that revealing these additional evaluations is 'safe' by increasing the amount of independence among values in $\hat{A}$ (while still restricting the verifier to evaluations outside of $\{0,1\}^s$), which would lead to a blowup in the degree of $\hat{A}$ that is proportional to the number of queries that the verifier wishes to make. For the low-degree test to work, however, the verifier *must* see more evaluations than the degree. In sum, this circularity is inherent. To solve this problem, we will design an 'algebraic' commitment scheme that is *transparent to low-degree tests*: the verifier can perform a low-degree test on the commitment itself (without the help of the prover), which will ensure access to a $\hat{A}$ that is low-degree. We discuss this further in Section 2.2.

(3) Sumcheck in zero knowledge. We need a sumcheck protocol where the prover's messages leak little information about f. The prior work in [BCFGRS16] achieves an IPCP for sumcheck that is 'weakly' zero knowledge: any verifier learns at most one evaluation of f for each query it makes to the PCP oracle. If the verifier could evaluate f by itself, as was the case in that paper, this guarantee would suffice for zero knowledge. In our setting, however, the verifier *cannot* evaluate f by itself because f is (necessarily) hidden behind the algebraic commitment.

One approach to compensate would be to further randomize $\hat{A}$ by letting $\hat{A}$ be a random extension of A of some well-chosen degree d. We are limited to d of polynomial size because the honest verifier's running time is $\Omega(d)$. But this means that a polynomial-time malicious verifier, participating in the protocol of [BCFGRS16] and making d^2 queries to the PCP oracle, could learn information about A.

We resolve this by relying on more algebraic techniques, achieving an IPCP for sumcheck with a much stronger zero knowledge guarantee (see Theorem 1.1): any malicious verifier that makes polynomially-many queries to the PCP oracle learns only a *single* evaluation of f. This suffices for zero knowledge in our setting: learning one evaluation of f implies learning only three evaluations of $\hat{A}$, which can be made 'safe' if $\hat{A}$ is chosen to be a random extension of A of high-enough degree. Our sumcheck protocol uses as building blocks both our algebraic commitment scheme and the [BCFGRS16] sumcheck; we summarize its construction in Section 2.3.

Remark 2.1. Kilian, Petrank, and Tardos [KPT97] construct PCPs for $\mathbf{NEXP}$ that are statistical zero knowledge, via a combinatorial construction that makes black-box use of the PCP Theorem. Our modification of the BFL protocol achieves a perfect zero knowledge IPCP via algebraic techniques, avoiding the use of the PCP Theorem.

2.2 Algebraic commitments from algebraic query complexity lower bounds

We describe how the sumcheck protocol can be used to construct an information-theoretic commitment scheme that is 'algebraic', in the IPCP model. (Namely, an algebraic *interactive locking scheme*; see Remark 2.2 below.) The prover commits to a message by sending to the verifier a PCP oracle that perfectly hides the message; subsequently, the prover can reveal positions of the message by engaging with the verifier in an Interactive Proof, whose soundness guarantees statistical binding. A key algebraic property that we rely on is that the commitment is 'transparent' to low-degree tests.

Committing to an element. We first consider the simple case of committing to a single element a in $\mathbb{F}$. Let k be a security parameter, and set $N := 2^k$. Suppose that the prover samples a random B in $\mathbb{F}^N$ such that $\sum_{i=1}^{N} B_i = a$, and sends B to the verifier as a commitment. Observe that any $N-1$ entries of B do not reveal any information about a, and so any verifier with oracle access to B that makes less than N queries cannot learn any information about a. However, as B is unstructured it is not clear how the prover can convince the verifier that $\sum_{i=1}^{N} B_i = a$.

Instead, we can consider imbuing B with additional structure by providing its low-degree extension. That is, the prover thinks of B as a function from $\{0,1\}^k$ to $\mathbb{F}$, and sends its unique multilinear extension $\hat{B}\colon \mathbb{F}^k \to \mathbb{F}$ to the verifier. Subsequently, the prover can reveal a to the verifier, and then engage in a sumcheck protocol for the claim "$\sum_{\vec{\beta}\in\{0,1\}^k} \hat{B}(\vec{\beta}) = a$" to establish the correctness of a. The soundness of the sumcheck protocol protects the verifier against cheating provers and hence guarantees that this scheme is binding.

However, giving B additional structure calls into question the hiding property of the scheme. Indeed, surprisingly a result of [JKRS09] shows that this new scheme is *not* hiding (in fields of characteristic different than 2): it holds that $\hat{B}(2^{-1},\ldots,2^{-1}) = a \cdot 2^{-k}$ for any choice of B, so the verifier can learn a with only a single query to $\hat{B}$!

Sending an extension of B has created a new problem: querying the extension outside of $\{0,1\}^k$, the verifier can learn information that may require many queries to B to compute. Indeed, this additional power is precisely what

underlies the soundness of the sumcheck protocol. To resolve this, we need to understand what the verifier can learn about B given some low-degree extension $\hat{B}$. This is precisely the setting of *algebraic query complexity* [AW09].

A natural approach is to let $\hat{B}$ be chosen uniformly at random from the set of degree-d extensions of B for some $d > 1$. It is not hard to see that if d is very large (say, $|\mathbb{F}|$) then 2^k queries are required to determine the summation of $\hat{B}$ on H^m. But we need d to be small to achieve soundness. A result of [JKRS09] shows that $d = 2$ suffices: given a random multiquadratic extension $\hat{B}$ of B, one needs 2^k queries to $\hat{B}$ to determine $\sum_{\vec{\beta} \in \{0,1\}^k} \hat{B}(\vec{\beta})$.

Committing to a polynomial. The prover in our zero knowledge protocols needs to commit not just to a single element but to the evaluation of an m-variate polynomial Q over $\mathbb{F}$ of degree d_Q. We extend our ideas to this setting.

Letting K be a subset of $\mathbb{F}$ of size $d_Q + 1$, the prover samples a random $B^{\vec{x}}$ in $\mathbb{F}^N$ such that $\sum_{i=1}^{N} B_i^{\vec{x}} = Q(\vec{x})$ for each $\vec{x} \in K^m$. We can view all of these strings as a single function $B \colon K^m \times \{0,1\}^k \to \mathbb{F}$, and as before we consider its unique low-degree extension $\hat{B} : \mathbb{F}^m \times \mathbb{F}^k \to \mathbb{F}$; viewed as a polynomial, $\hat{B}(\vec{X}, \vec{Y})$ has degree at most d_Q in $\vec{X}$ and is multilinear in $\vec{Y}$. Observe that since $\sum_{\vec{\beta} \in \{0,1\}^k} \hat{B}(\vec{X}, \vec{\beta})$ is a polynomial of individual degree d_Q that agrees with Q on K^m, it must equal Q. The binding property of the commitment scheme is clear: the prover can decommit to $Q(\vec{\alpha})$ for any $\vec{\alpha} \in \mathbb{F}^m$ by using the sumcheck protocol as before. We are left to argue the hiding property.

It is not difficult to see that we run into the same issue as in the single-value case: we have $\hat{B}(\vec{\alpha}, 2^{-1}, \ldots, 2^{-1}) = Q(\vec{\alpha}) \cdot 2^{-k}$ for any $\vec{\alpha} \in \mathbb{F}^m$. We resolve this by again choosing a *random* extension $\hat{B}$ of degree $d > 1$ in $\vec{Y}$ (and degree d_Q in $\vec{X}$). Yet, arguing the hiding property now requires a *stronger* statement than the one proved in [JKRS09]. Not only do we need to know that the verifier cannot determine $Q(\vec{\alpha})$ for a particular $\vec{\alpha} \in \mathbb{F}^m$, but we need to know that the verifier cannot determine $Q(\vec{\alpha})$ for *any* $\vec{\alpha} \in \mathbb{F}^m$, or even *any linear combination of any such values*. We prove that this stronger guarantee holds in the same parameter regime: if $d > 1$ then 2^k queries are both necessary and sufficient.

Transparency to low-degree tests. Recall that a key algebraic property we required from our commitment scheme is that the verifier can perform a low-degree test on the committed polynomial without the assistance of the prover. Our commitment scheme naturally has this property. If the PCP oracle $\tilde{B} \colon \mathbb{F}^m \times \mathbb{F}^k \to \mathbb{F}$ is low-degree, then it is a commitment to the low-degree $Q \colon \mathbb{F}^m \to \mathbb{F}$ defined as $Q(\vec{X}) := \sum_{\vec{\beta} \in \{0,1\}^k} \tilde{B}(\vec{X}, \vec{\beta})$. In fact, even if $\tilde{B} \colon \mathbb{F}^m \times \mathbb{F}^k \to \mathbb{F}$ is merely *close* to a low-degree $\hat{B} \colon \mathbb{F}^m \times \mathbb{F}^k \to \mathbb{F}$, then we can still regard $\tilde{B}$ as a commitment to the low-degree $Q \colon \mathbb{F}^m \to \mathbb{F}$ defined as $Q(\vec{X}) := \sum_{\vec{\beta} \in \{0,1\}^k} \hat{B}(\vec{X}, \vec{\beta})$, because the verifier can check claims of the form "$Q(\vec{\alpha}) = a$" by obtaining the value of $\hat{B}$ it needs at the end of the sumcheck protocol via self-correction on $\tilde{B}$.

Beyond the boolean hypercube. For efficiency reasons analogous to those in [BFLS91; GKR15], instead of viewing B as a function from $K^m \times \{0,1\}^k$ to $\mathbb{F}$, we view B as a function from $K^m \times H^{k'}$ to $\mathbb{F}$ for a subset H of $\mathbb{F}$ of size $\mathbb{F}^{\Omega(1)}$ and $k' := \log N / \log |H|$. This requires us to extend our claims about the algebraic query complexity of polynomial summation to arbitrary sets H. We show that if $d > 2(|H| - 1)$, then $|H|^{k'} = N$ queries are necessary to determine $Q(\vec{\alpha})$ for any $\vec{\alpha}$ (or any linear combination of these). See Section 5 for details.

Decommitting in zero knowledge. To use our commitment scheme in zero knowledge protocols, we must ensure that, in the decommitment phase, the verifier cannot learn any information beyond the value $a := Q(\vec{\alpha})$ for a chosen $\vec{\alpha}$. To decommit, the prover sends the value a and has to convince the verifier that the claim "$\sum_{\vec{\beta} \in \{0,1\}^k} \hat{B}(\vec{\alpha}, \vec{\beta}) = a$" is true. However, if the prover and verifier simply run the sumcheck protocol on this claim, the prover leaks partial sums $\sum_{\vec{\beta} \in \{0,1\}^{k-i}} \hat{B}(\vec{\alpha}, c_1, \ldots, c_i, \vec{\beta})$ for $c_1, \ldots, c_i \in \mathbb{F}$ chosen by the verifier, which could reveal additional information about Q. Instead, the prover and verifier run on this claim the IPCP for sumcheck of [BCFGRS16], whose 'weak' zero knowledge guarantee ensures that this cannot happen. (Thus, in addition to the commitment, the honest prover also sends the evaluation of a random low-degree polynomial as required by the IPCP for sumcheck of [BCFGRS16].)

Remark 2.2 (comparison with [GIMS10])**.** Goyal, Ishai, Mahmoody, and Sahai [GIMS10] define and construct *interactive locking schemes*, information-theoretic commitment schemes in the IPCP model. Their scheme is combinatorial, and we do not know how to use it in our setting (it is not clear how to low-degree test the committed message without disrupting zero knowledge). Putting this difference aside, their construction and our construction are incomparable. On the one hand, we achieve perfect hiding while they only achieve statistical hiding. On the other hand, their scheme is 'oracle efficient' (any query to the oracle can be computed statelessly in polynomial time) while our scheme is not.

2.3 A zero knowledge sumcheck protocol

We summarize the ideas behind our main result, a zero knowledge sumcheck protocol (see Theorem 1.1). This result not only enables us to modify the BFL protocol to achieve zero knowledge (as discussed above), but also to modify the Shamir and GKR protocols to achieve zero knowledge (as discussed below). The two building blocks underlying our sumcheck protocol are our algebraic commitments (see Section 2.2 above) and the IPCP for sumcheck of [BCFGRS16]. We now cover necessary background and then describe our protocol.

Previous sumcheck protocols. The sumcheck protocol [LFKN92] is an IP for claims of the form "$\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) = 0$", where H is a subset of a finite field $\mathbb{F}$ and F is an m-variate polynomial over $\mathbb{F}$ of small individual degree. The protocol has m rounds: in round i, the prover sends the univariate polynomial $g_i(X_i) := \sum_{\vec{\alpha} \in H^{m-i}} F(c_1, \ldots, c_{i-1}, X_i, \vec{\alpha})$; the verifier checks that $\sum_{\alpha_i \in H} g_i(\alpha_i) = g_{i-1}(c_{i-1})$ and replies with a uniformly random challenge $c_i \in \mathbb{F}$. After round m, the verifier outputs the claim "$F(c_1, \ldots, c_m) = g_m(c_1, \ldots, c_m)$". If F is of sufficiently low degree and does not sum to a over the space, then the output claim is false with high probability. Note that the verifier does not need access to F.

The IPCP for sumcheck of [BCFGRS16] modifies the above protocol as follows. The prover first sends a PCP oracle that equals the evaluation of a random 'masking' polynomial R; the verifier checks that R is (close to) low degree. After that the prover and verifier conduct an Interactive Proof. The prover sends $z \in \mathbb{F}$ that allegedly equals $\sum_{\vec{\alpha} \in H^m} R(\vec{\alpha})$, and the verifier responds with a uniformly random challenge $\rho \in \mathbb{F}^*$. The prover and verifier now run the (standard) sumcheck protocol to reduce the claim "$\sum_{\vec{\alpha} \in H^m} \rho F(\vec{\alpha}) + R(\vec{\alpha}) = \rho a + z$" to a claim "$\rho F(\vec{c}) + R(\vec{c}) = b$" for random $\vec{c} \in \mathbb{F}^m$. The verifier queries R at $\vec{c}$ and then outputs the claim "$F(\vec{c}) = \frac{b - R(\vec{c})}{\rho}$". If $\sum_{\vec{\alpha} \in H^m} F(\vec{\alpha}) \neq a$ then with high probability over the choice of ρ and the verifier's messages in the sumcheck protocol, this claim will be false.

A key observation is that if the verifier makes no queries to R, then the prover's messages are identically distributed to the sumcheck protocol applied to a uniformly random polynomial Q. When the verifier does make queries to R, simulating the resulting conditional distribution involves techniques from Algebraic Complexity Theory, as shown in [BCFGRS16]. Given Q, the verifier's queries to $R(\vec{\alpha})$ for $\vec{\alpha} \in \mathbb{F}^m$ are identically distributed to $Q(\vec{\alpha}) - \rho F(\vec{\alpha})$. Thus the simulator need only make at most one query to F for every query to R. That is, any verifier making q queries to R learns no more than it would learn by making q queries to F alone.

As discussed, this zero knowledge guarantee does not suffice for the applications that we consider: when a sumcheck protocol is used as a subroutine of another protocol, F may itself be recursively defined in terms of large sums which the verifier cannot evaluate on its own. The verifier does, however, have oracle access to R, and so can learn enough information about F to break zero knowledge.

Our sumcheck protocol. The zero knowledge guarantee that we aim for is the following: any polynomial-time verifier learns no more than it would by making *one* query to F, regardless of its number of queries to the PCP oracle.

The main idea to achieve this guarantee is the following. The prover sends a PCP oracle that is an *algebraic commitment* Z to the aforementioned masking polynomial R. Then, as before, the prover and verifier run the sumcheck protocol to reduce the claim "$\sum_{\vec{\alpha} \in H^m} \rho F(\vec{\alpha}) + R(\vec{\alpha}) = \rho a + z$" to a claim "$\rho F(\vec{c}) + R(\vec{c}) = b$" for random $\vec{c} \in \mathbb{F}^m$.

We now face two problems. First, the verifier cannot simply query R at $\vec{c}$ and then output the claim "$F(\vec{c}) = \frac{b - R(\vec{c})}{\rho}$", since the verifier only has oracle access to the commitment Z of R. Second, the prover could cheat the verifier by having Z be a commitment to an R that is far from low degree, which allows cheating in the sumcheck protocol.

The first problem is addressed by the fact that our algebraic commitment scheme has a decommitment sub-protocol that is zero knowledge: the prover can reveal $R(\vec{c})$ in such a way that no other values about R are also revealed as a side-effect. As discussed, this relies on the protocol of [BCFGRS16], used a subroutine (for the second time).

The second problem is taken care of by the fact that our algebraic commitment scheme is 'transparent' to low-degree tests: the verifier simply performs a low-degree test on Z, which by self-correction gives the verifier oracle access to a low-degree Z' that *is* a commitment to a low-degree R.

Overall, the only value that a malicious verifier can learn is $F(\vec{c})$ for $\vec{c} \in \mathbb{F}^m$ of its choice.

Remark 2.3. Our sumcheck protocol 'leaks' a single evaluation of F. We believe that this limitation is inherent: the honest verifier always outputs a true claim about one evaluation of F, which it cannot do without learning that evaluation. Either way, this guarantee is strong enough for applications: we ensure that learning a single evaluation of F does not harm zero knowledge, either because it carries no information or because the verifier can evaluate F itself.

2.4 Challenges: handling recursion

We have so far discussed the ideas behind our main result (a zero knowledge sumcheck protocol) and how to use it to achieve a natural zero knowledge analogue of the classical MIP/PCP construction for $\mathbf{NEXP}$ [BFL91; BFLS91]. Other applications require additional ideas to overcome challenges that arise when the sumcheck protocol is used *recursively*.

Shamir's protocol. Consider the goal of achieving zero knowledge for Shamir's protocol for $\mathbf{PSPACE}$ [Sha92]. This protocol reduces checking any $\mathbf{PSPACE}$ computation to checking that:

$$\sum_{x_1\in\{0,1\}} \prod_{x_2\in\{0,1\}} \cdots \sum_{x_{n-1}\in\{0,1\}} \prod_{x_n\in\{0,1\}} \hat{\phi}(x_1,\ldots,x_n) = 0$$

where $\hat{\phi}$ is the (efficiently computable) arithmetization over a finite field $\mathbb{F}$ of a certain boolean formula ϕ. Since Shamir's protocol is similar to the sumcheck protocol, a natural starting point would be to try to merely adapt the techniques that 'worked' in the case of the sumcheck protocol. However, the similarity between the two protocols is only superficial (e.g., it lacks the useful linear structure present in the sumcheck protocol). An accurate way to compare the two is to view Shamir's protocol as a *recursive* application of the sumcheck protocol, as shown by Meir [Mei13].

For example, the TQBF problem is *downward self-reducible* [TV07]: for $i \in \{1,\ldots,n\}$ let

$$G_i(X_1,\ldots,X_i) := \sum_{x_{i+1}\in\{0,1\}} \prod_{x_{i+2}\in\{0,1\}} \cdots \sum_{x_{n-1}\in\{0,1\}} \prod_{x_n\in\{0,1\}} \hat{\phi}(X_1,\ldots,X_i,x_{i+1},\ldots,x_n) \ .$$

From this one obtains the recurrence

$$G_i(X_1,\ldots,X_i) = \sum_{x_{i+1}\in\{0,1\}} G_{i+2}(X_1,\ldots,X_i,x_{i+1},0)\cdot G_{i+2}(X_1,\ldots,X_i,x_{i+1},1) \ . \tag{1}$$

In other words, with oracle access to G_{i+2}, one can use a (small) sumcheck to compute G_i. This suggests a recursive approach: if we could check evaluations of G_{i+2} in zero knowledge, then maybe we could use this as a subprotocol to check evaluations of G_i also in zero knowledge.

GKR's protocol. The recursive structure is perhaps more evident in the doubly-efficient Interactive Proof of Goldwasser, Kalai, and Rothblum [GKR15] ('GKR protocol'). Its barebones sub-protocol checks a more complex arithmetic expression: the output of a layered arithmetic circuit. Fix an input x to the circuit, and let $V_i(j)\colon [S] \to \mathbb{F}$ be the value of the j-th gate in layer i (S is the number of gates in a layer). For some subset $H \subseteq \mathbb{F}$ and sufficiently large m, one views V_i as a function from H^m to $\mathbb{F}$ by imposing some ordering on H^m. One can relate V_{i-1} to V_i as follows:

$$V_{i-1}(\vec{z}) = \sum_{\vec{\omega}_1,\vec{\omega}_2\in H^m} \mathrm{add}_i(\vec{z},\vec{\omega}_1,\vec{\omega}_2)\cdot\big(V_i(\vec{\omega}_1)+V_i(\vec{\omega}_2)\big) + \mathrm{mul}_i(\vec{z},\vec{\omega}_1,\vec{\omega}_2)\cdot\big(V_i(\vec{\omega}_1)\cdot V_i(\vec{\omega}_2)\big) \tag{2}$$

where $\mathrm{add}_i(\vec{z},\vec{\omega}_1,\vec{\omega}_2)$ is 1 if the $\vec{z}$-th gate in layer $i-1$ is an addition gate whose inputs are the $\vec{\omega}_1$-th and $\vec{\omega}_2$-th gates in layer i, and mul_i is defined similarly for multiplication gates.

We again see a recursive structure: a function defined as the summation over some product space of a polynomial whose terms are functions of the same form; this allows to check V_{i-1} given a protocol for checking V_i.

The use of recursion in the GKR protocol is even more involved: the barebones protocol relies on the verifier having oracle access to low-degree extensions of add_i and mul_i. For very restricted classes of circuits, the verifier can efficiently 'implement' these oracles; however, for the class of circuits that is ultimately supported by the protocol this requires a further sub-protocol that delegates the evaluation of these oracles to the prover, and this is done by *composing* multiple instances of the GKR protocol. To achieve zero knowledge we also have to tackle this form of recursion.

The leakage of recursion. By now the central role of recursion in applications of the sumcheck protocol is clear. There are two main sources of leakage that we need to overcome in order to achieve zero knowledge in such applications.

1. Checking evaluations of G_{i+2}, V_i, or add_i and mul_i, even in zero knowledge, leaks the evaluations themselves. The verifier, however, is not able to compute these itself (else it would not need to delegate), which means that information is leaked.

2. The number of claims can grow exponentially: a claim about G_i (resp. V_{i-1}) is reduced to *two* claims about G_{i+2} (resp. V_i). There are standard techniques that leverage interaction to reduce multiple claims about a low-degree polynomial to a single one, but we need to replace these with zero knowledge equivalents.

We tackle both issues by devising a general framework that captures their shared algebraic structure, solving these problems within this framework, and then recovering the protocols of Shamir and GKR as special cases.

2.5 Sum-product circuits

We introduce the notion of *sum-product circuits* and show that the sumcheck protocol naturally gives rise to algebraic Interactive Proofs for checking the value of such circuits. We then explain how to achieve zero knowledge variants of these by building on the techniques discussed in Section 2.1. We recover zero knowledge variants of the protocols of Shamir and GKR as special cases of this approach.

Sum-product circuits are an abstract way of encoding 'sum-product expressions'. A sum-product expression is either a polynomial over some finite field $\mathbb{F}$ represented by a small arithmetic circuit or a polynomial of the form

$$\sum_{\vec{\beta} \in H^m} C\big(\vec{X}, \vec{\beta}, P_1(\vec{X}, \vec{\beta}), \ldots, P_n(\vec{X}, \vec{\beta})\big) \tag{3}$$

where C is a low-degree 'combiner' polynomial represented by a small arithmetic circuit, and $P_1, \ldots, P_n$ are sum-product expressions. Both Equation 1 (for Shamir's protocol) and Equation 2 (for GKR's protocol) are of this form.

Like a standard arithmetic circuit, a sum-product circuit is a directed acyclic graph associated with a field $\mathbb{F}$ in which we associate to each vertex a *value*, which in our case is the sum-product expression that it computes. Each internal vertex is labeled by a combiner polynomial, and there is an edge from u to v if the sum-product expression of v appears in that of u. For example, the above expression would correspond to a vertex labeled with C, with outgoing edges to the vertices corresponding to $P_1, \ldots, P_n$. An input to the circuit is a labeling of the leaf vertices with small arithmetic circuits. We now spell this out in a little more detail.

Definition 2.4 (Informal version of Definition 8.8)**.** *A* **sum-product circuit** $\mathcal{C}$ *is a rooted directed acyclic graph where each internal vertex is labeled with an arithmetic circuit C_v over a finite field $\mathbb{F}$. An* input $\mathtt{x}$ *to $\mathcal{C}$ labels each leaf v with a polynomial $\mathtt{x}_v$ over $\mathbb{F}$. The value of a vertex v on input $\mathtt{x}$ is a multivariate polynomial $v[\mathtt{x}]$ over $\mathbb{F}$ defined as follows: if v is a leaf vertex then $v[\mathtt{x}]$ equals $\mathtt{x}_v$; if instead v is an internal vertex then, for a chosen integer m,*

$$v[\mathtt{x}](\vec{X}) := \sum_{\vec{\beta} \in H^m} C_v\big(\vec{X}, \vec{\beta}, u_1[\mathtt{x}](\vec{X}, \vec{\beta}), \ldots, u_t[\mathtt{x}](\vec{X}, \vec{\beta})\big) \ . \tag{4}$$

The **value** *of $\mathcal{C}$ on input $\mathtt{x}$ is denoted $\mathcal{C}[\mathtt{x}]$ and equals the value of the root vertex r (and we require that $\mathcal{C}[\mathtt{x}] \in \mathbb{F}$).*

We next describe an Interactive Proof that works for any sum-product circuit. The protocols of Shamir [Sha92] and of GKR [GKR15] can be viewed as this protocol applied to specific sum-product circuits (computing G_0 and V_0 respectively). After that, we explain how to modify the Interactive Proof to obtain a corresponding zero knowledge IPCP for any sum-product circuit, which allows us to derive our zero knowledge variants of these two protocols.

A significant advantage of working with sum-product circuits is that they are easy to compose. For example, we can view the composition of the GKR protocol with itself as a *composition of sum-product circuits*. We can then apply our zero knowledge IPCP to the resulting circuit and directly obtain a zero knowledge analogue of the full GKR protocol.

2.5.1 Delegating the evaluation of a sum-product circuit

We explain how to use the sumcheck protocol to obtain an Interactive Proof for checking the value of a sum-product circuit. The protocol is recursively defined: to prove that $\mathcal{C}[\mathtt{x}] = a$ (i.e., that $r[\mathtt{x}] = a$), it suffices to show that the values of r's children $u_1, \ldots, u_t$ satisfy Equation 4 where the left-hand side is a. The sumcheck protocol interactively reduces this claim to a new claim "$C_v\big(\vec{c}, u_1[\mathtt{x}](\vec{c}), \ldots, u_t[\mathtt{x}](\vec{c})\big) = b$" for $\vec{c} \in \mathbb{F}^m$ chosen uniformly at random by the verifier and $b \in \mathbb{F}$ chosen by the prover. The prover sends $h_1 := u_1[\mathtt{x}](\vec{c}), \ldots, h_t := u_t[\mathtt{x}](\vec{c})$, reducing this new claim

to the set of claims "$h_i = u_i[\mathtt{x}](\vec{c})$" for $i = 1, \ldots, t$ and "$C_v(\vec{c}, h_1, \ldots, h_t) = b$". The latter can be checked by the verifier directly and the rest can be recursively checked via the same procedure. Eventually the protocol reaches the leaf vertices, which are labeled with small arithmetic circuits that the verifier can evaluate on its own.

One technicality is that, as defined, the degree of the polynomial at a vertex may be exponentially large, and so the prover would have to send exponentially-large messages in the sumcheck protocol. To avoid this, we use a well-known interactive sub-protocol for *degree reduction* [She92; GKR15]. Since for all $\vec{x}$ the value $v[\mathtt{x}](\vec{x})$ depends only on $u_1[\mathtt{x}](\vec{x}, \vec{\beta}), \ldots, u_t[\mathtt{x}](\vec{x}, \vec{\beta})$ for $\vec{\beta} \in H^m$, we can safely replace each $u_i[\mathtt{x}]$ with the unique degree-$(|H| - 1)$ extension $\hat{u}_i[\mathtt{x}]$ of its evaluation over H^m. The degree of the summand in Equation 4 is now at most $\delta|H|$, where δ is the *total* degree of C_v. Now that the sumcheck protocol is only invoked on low-degree polynomials, efficiency is recovered.

Another technicality is that since a sum-product circuit is a directed acyclic graph (as opposed to a tree), it is possible that a single vertex v will have many claims about it. If each such claim reduces to many claims about other vertices, the number of claims to check could grow exponentially. This is in fact the case in both Shamir's and GKR's protocols. To avoid this blowup, the verifier checks a random linear combination of the claims about each vertex v. It is not difficult to see that soundness is preserved, and the number of claims per vertex is reduced to one.

2.5.2 Achieving zero knowledge

The Interactive Proof for sum-product circuits that we have described above is not zero knowledge. First, the sumcheck protocol, which is used to reduce claims about parent vertices to claims about child vertices, leaks information in the form of partial sums of the summand polynomial, as usual. Second, in order to reduce a claim about the root to claims about its children, the prover must provide evaluations of the polynomials of the children. These may be hard for the verifier to compute (indeed, if the verifier could compute both of these on its own then there would be no need to recurse). We use the ideas discussed in Section 2.1 to resolve both of these issues, obtaining a zero knowledge variant in the IPCP model (where the honest prover sends a random low-degree polynomial as the oracle).

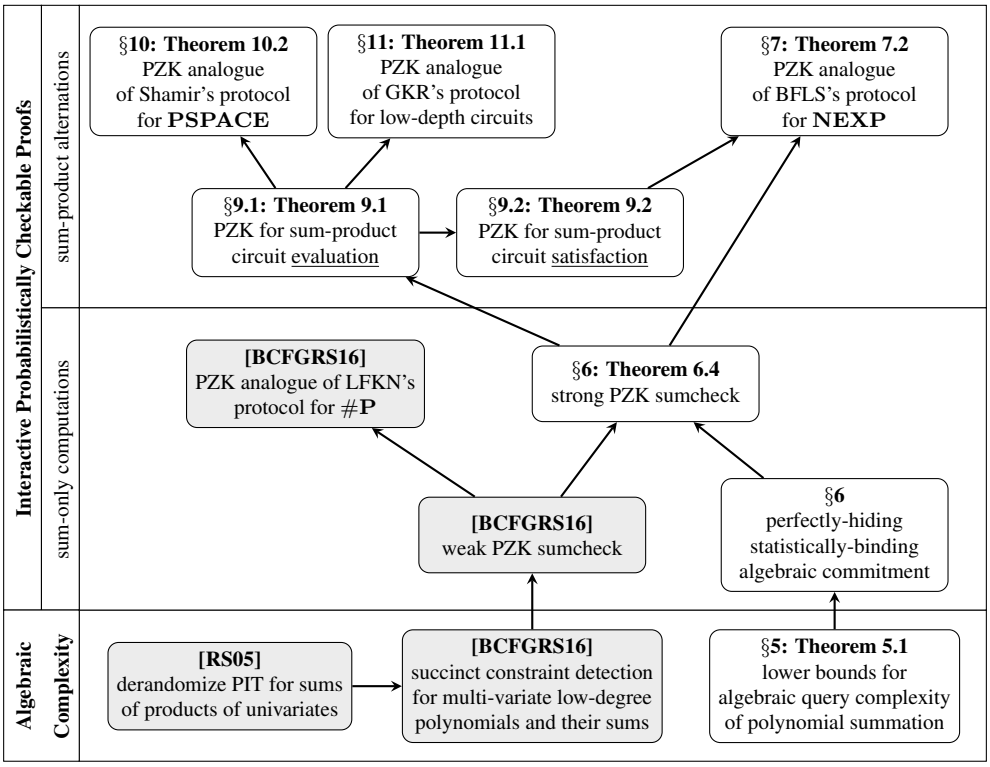
We resolve the first issue by using our zero knowledge sumcheck protocol. Its zero knowledge guarantee states that the protocol reveals only one value of the summand function, which can be computed via one query to each of the $\hat{u}_i[\mathtt{x}]$, which are precisely the h_i's sent by the prover. We are left to ensure that h_i's do not leak information.

As in our modification of the BFL protocol, rather than taking the unique degree-$(|H| - 1)$ extension $\hat{v}[\mathtt{x}](\vec{X})$ of $v[\mathtt{x}](\vec{X})$, we will instead take a *random* degree-$(|H| + \delta)$ extension $\dot{v}[\mathtt{x}]$, where δ depends only on the circuit structure (in all of our protocols, δ is a small constant). This ensures that the few evaluations actually revealed by the prover are uniformly random in $\mathbb{F}$. The prover sends, for each vertex v, the evaluation of a random polynomial R_v, which defines the random low-degree extension as $\dot{v}[\mathtt{x}](\vec{X}) := \hat{v}[\mathtt{x}](\vec{X}) + \mathbb{Z}_{H^m}(\vec{X}) \cdot R_v(\vec{X})$ where $\mathbb{Z}_{H^m}$ is a degree-$|H|$ polynomial that is zero on H^m and nonzero on $(\mathbb{F} - H)^m$. The prover cannot simply send R_v, however, because the verifier could then query it in order to 'derandomize' $\dot{v}[\mathtt{x}]$. Instead, the prover sends a commitment to R_v using our algebraic commitment scheme. The decommitment is performed 'implicitly' during the sumcheck for vertex v. See Section 9.1 for details.

Finally, recall that in order to avoid a blowup in the number of claims we have to check, the verifier checks a random linear combination of the claims about any given vertex; this is a linear operation. Also, to avoid a blowup in the degree, we take the low-degree extension, which is also a linear operation. Both of these operations are 'compatible' with sumcheck, and thus zero knowledge is straightforwardly maintained.

3 Roadmap

After providing formal definitions in Section 4, the rest of the paper is organized as summarized by the table below. The shaded boxes denote some previous results that we rely on.



4 Preliminaries

4.1 Basic notations

For $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. For $m, n \in \mathbb{N}$ we denote by $m + [n]$ the set $\{m+1, \ldots, m+n\}$. For a set X, $n \in \mathbb{N}$, $I \subseteq [n]$, and $\vec{x} \in X^n$, we denote by $\vec{x}_I$ the vector $\left(x_i\right)_{i \in I}$ that is $\vec{x}$ restricted to the coordinates in I.

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. Arithmetic operations over $\mathbb{F}_q$ take time $\operatorname{polylog} q$ and space $O(\log q)$.

Polynomials. We denote by $\mathbb{F}[X_{1,\ldots,m}]$ the ring of polynomials in m variables over $\mathbb{F}$. Given a polynomial P in $\mathbb{F}[X_{1,\ldots,m}]$, $\deg_{X_i}(P)$ is the degree of P in the variable X_i. The *individual degree* of a polynomial is its maximum degree in any variable, $\max_{1 \le i \le m} \deg_{X_i}(P)$; we always refer to the individual degree unless otherwise specified. We denote by $\mathbb{F}[X^{\le d}_{1,\ldots,m}]$ the subspace consisting of $P \in \mathbb{F}[X_{1,\ldots,m}]$ with individual degree at most d.

Languages and relations. We denote by $\mathscr{L}$ a language consisting of *instances* $\mathtt{x}$, and 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 $T(n) \ge n$.

Low-degree extensions. Let $\mathbb{F}$ be a finite filed, H a subset of $\mathbb{F}$, and m a positive integer. The *low-degree extension* (LDE) of a function $f\colon H^m \to \mathbb{F}$ is denoted $\hat{f}$ and is the unique polynomial in $\mathbb{F}[X^{\le |H|-1}_{1,\ldots,m}]$ that agrees with f on H^m. In particular, $\hat{f}\colon \mathbb{F}^m \to \mathbb{F}$ is defined as follows:

$$\hat{f}(\vec{X}) := \sum_{\vec{\beta} \in H^m} I_{H^m}(\vec{X}, \vec{\beta}) \cdot f(\vec{\beta}) \ ,$$

where $I_{H^m}(\vec{X}, \vec{Y}) := \prod_{i=1}^{m} \sum_{\omega \in H} \prod_{\gamma \in H \setminus \{\omega\}} \frac{(X_i - \gamma)(Y_i - \gamma)}{(\omega - \gamma)^2}$ is the unique polynomial in $\mathbb{F}[X^{\le |H|-1}_{1,\ldots,m}]$ such that, for all $(\vec{\alpha}, \vec{\beta}) \in H^m \times H^m$, $I_{H^m}(\vec{\alpha}, \vec{\beta})$ equals 1 when $\vec{\alpha} = \vec{\beta}$ and equals 0 otherwise. Note that $I_{H^m}(\vec{X}, \vec{Y})$ can be generated and evaluated in time $\operatorname{poly}(|H|, m, \log|\mathbb{F}|)$ and space $O(\log|\mathbb{F}| + \log m)$, so $\hat{f}(\vec{\alpha})$ can be evaluated in time $|H|^m \cdot \operatorname{poly}(|H|, m, \log|\mathbb{F}|)$ and space $O(m \cdot \log|\mathbb{F}|)$.

4.2 Sampling partial sums of random low-degree polynomials

Let $\mathbb{F}$ be a finite field, m, d positive integers, and H a subset of $\mathbb{F}$, and recall that $\mathbb{F}[X^{\le d}_{1,\ldots,m}]$ is the subspace of $\mathbb{F}[X_{1,\ldots,m}]$ consisting of those polynomials with individual degrees at most d. Given $Q \in \mathbb{F}[X^{\le d}_{1,\ldots,m}]$ and $\vec{\alpha} \in \mathbb{F}^{\le m}$ (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 prefix of the variables is the sum of the values obtained by letting the remaining variables range over H.

In Section 6 we rely on the fact, formally stated below and proved in [BCFGRS16], that one can efficiently sample the distribution $R(\vec{\alpha})$, where R is uniformly random in $\mathbb{F}[X^{\le d}_{1,\ldots,m}]$ and $\vec{\alpha} \in \mathbb{F}^{\le m}$ is fixed, *even conditioned on any polynomial number of (consistent) values for* $R(\vec{\alpha}_1), \ldots, R(\vec{\alpha}_\ell)$ (with $\vec{\alpha}_1, \ldots, \vec{\alpha}_\ell \in \mathbb{F}^{\le m}$). More precisely, the sampling algorithm runs in time that is only $\operatorname{poly}(\log|\mathbb{F}|, m, d, |H|, \ell)$, which is much faster than the trivial running time of $\Omega(d^m)$ achieved by sampling R explicitly. This "succinct" sampling follows from the notion of *succinct constraint detection* studied in [BCFGRS16] for the case of partial sums of low-degree polynomials.

Corollary 4.1 ([BCFGRS16])**.** *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 $S = \{(\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)\} \subseteq \mathbb{F}^{\le m} \times \mathbb{F}$, and $(\alpha, \beta) \in \mathbb{F}^{\le m} \times \mathbb{F}$,*

$$\Pr\left[\mathcal{A}(\mathbb{F}, m, d, H, S, \alpha) = \beta\right] = \Pr_{R \leftarrow \mathbb{F}[X^{\le d}_{1,\ldots,m}]}\left[R(\alpha) = \beta \;\middle|\; \begin{array}{c} R(\alpha_1) = \beta_1 \\ \vdots \\ R(\alpha_\ell) = \beta_\ell \end{array} \right] \ .$$

Moreover $\mathcal{A}$ runs in time $m(d\ell|H| + d^3\ell^3) \cdot \mathrm{poly}(\log|\mathbb{F}|) = \ell^3 \cdot \mathrm{poly}(m, d, |H|, \log|\mathbb{F}|)$.

4.3 Interactive probabilistically checkable proofs

An *Interactive Probabilistically Checkable Proof* (Interactive PCP, IPCP) [KR08] is a Probabilistically Checkable Proof [BFLS91; FGLSS91; AS98; ALMSS98] followed by an Interactive Proof [Bab85; GMR89]. 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 k-round interactive proof. (Though note that [BCFGRS16] counts the PCP round towards round complexity.) 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 language $\mathscr{L}$ (resp., relation $\mathscr{R}$) belongs to the complexity class $\mathbf{IPCP}[\varepsilon, \mathsf{k}, \mathsf{l}, \mathsf{q}]$ if there is an IPCP system for $\mathscr{L}$ (resp., $\mathscr{R}$) in which: (1) the soundness error is $\varepsilon(n)$; (2) the number of rounds is at most $\mathsf{k}(n)$; (3) the proof length is at most $\mathsf{l}(n)$; (4) the query complexity is at most $\mathsf{q}(n)$. We sometimes also specify the time and/or space complexity of the (honest) prover algorithm and/or (honest) verifier algorithm.

Finally, an IPCP is *non-adaptive* if the verifier queries are non-adaptive, i.e., the queried locations depend only on the verifier's inputs; it is *public-coin* if each verifier message is chosen uniformly and independently at random, and all of the verifier queries happen after receiving the last prover message. *All of the IPCPs discussed in this paper are both non-adaptive and public-coin.*

4.4 Zero knowledge for Interactive PCPs

We define the notion of zero knowledge for IPCPs that we consider: *perfect zero knowledge via straightline simulators*. This notion is quite strong not only because it unconditionally guarantees perfect simulation of the verifier's view but also because straightline simulation typically implies desirable properties. We first provide context and then 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 definition below considers the case of perfect simulation via universal simulators against verifiers making a bounded number of queries to the proof oracle.

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 (our setting), the natural definition would allow a blackbox simulator to rewind the verifier *and also* to adaptively answer oracle queries. The definition below, however, considers 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 perfect zero knowledge via straightline simulators for IPCPs [GIMS10].

Definition 4.2. *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 IPCP protocol with $B(y)$, i.e., the random variable $(x, r, s_1, \ldots, s_n, t_1, \ldots, t_m)$ where x is A's input, r is A's randomness, $s_1, \ldots, s_n$ are B's messages, and $t_1, \ldots, t_m$ are the answers to A's queries to the proof oracle sent by B.*

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 IPCPs, where the simulator also has to answer oracle queries by the verifier. Note that since we consider the notion of perfect zero knowledge, the definition of straightline simulation needs to allow the efficient simulator to work even with inefficient verifiers [GIMS10].

Definition 4.3. *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.)*

Definition 4.4. *An IPCP system (P, V) for a relation $\mathscr{R}$ is perfect zero knowledge (via straightline simulators) against unbounded queries (resp., against query bound b) with simulator overhead $s\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ 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* View $\langle P(\mathtt{x}, \mathtt{w}), \tilde{V}(\mathtt{x})\rangle$ *are identically distributed. Moreover, S must run in time $O(s(|\mathtt{x}|, \mathsf{q}_{\tilde{V}}(|\mathtt{x}|)))$, where $\mathsf{q}_{\tilde{V}}(\cdot)$ is $\tilde{V}$'s query complexity.*

The case of a language $\mathscr{L}$ is similar: the quantification is for all $\mathtt{x} \in \mathscr{L}$ and the view to simulate is View $\langle P(\mathtt{x}), \tilde{V}(\mathtt{x})\rangle$.

Remark 4.5. Throughout this paper, an algorithm is b-***query*** if it makes strictly fewer than b queries to its oracle. This is because all of our results will be of a 'query threshold' character, i.e. if the verifier makes b queries it learns some information, but any verifier making strictly fewer queries learns nothing.

Remark 4.6. The standard definition of zero knowledge allows the simulator overhead s to be any fixed polynomial.

Remark 4.7. The definition above places a strict bound on the running time of the simulator. This is in contrast to most zero knowledge results, which can only bound its *expected* running time.

We say that a language $\mathscr{L}$ (resp., relation $\mathscr{R}$) belongs to the complexity class $\mathbf{PZK\text{-}IPCP}[\varepsilon, \mathsf{k}, \mathsf{l}, \mathsf{q}, \mathsf{b}, s]$ if there is an IPCP system for $\mathscr{L}$ (resp., $\mathscr{R}$), with the corresponding parameters, that is perfect zero knowledge with query bound b; also, it belongs to the complexity class $\mathbf{PZK\text{-}IPCP}[\varepsilon, \mathsf{k}, \mathsf{l}, \mathsf{q}, *, s]$ if the same is true with unbounded queries. In this paper we only consider zero knowledge against bounded queries. (Note that, even in this case, one can 'cover' all polynomial-time malicious verifiers by setting b to be superpolynomial in the input size.)

Remark 4.8. Kalai and Raz [KR08] give a general transformation for IPCPs that reduces the verifier's query complexity q to 1. The transformation preserves our zero knowledge guarantee, with a small increase in the simulator overhead.

4.5 Sumcheck protocol and its zero knowledge variant

The sumcheck protocol [LFKN92] is a fundamental building block of numerous results in complexity theory and cryptography. We rely on it in Section 6, so we briefly review it here. The protocol consists of an Interactive Proof for a claim of the form "$\sum_{\alpha_1,\ldots,\alpha_m \in H} F(\alpha_1, \ldots, \alpha_m) = a$", where F is an m-variate polynomial of individual degree d with coefficients in a finite field $\mathbb{F}$, H is a subset of $\mathbb{F}$, and a is an element of $\mathbb{F}$. The prover and verifier receive $(\mathbb{F}, m, d, H, a)$ as input; in addition, the prover receives F as input while the verifier has only oracle access to F. In the i-th round, the prover sends the univariate polynomial $F_i(X) := \sum_{\alpha_{i+1},\ldots,\alpha_m \in H} F(c_1, \ldots, c_{i-1}, X, \alpha_{i+1}, \ldots, \alpha_m)$, and the verifier replies with a uniformly random element $c_i \in \mathbb{F}$ and checks that $F_{i-1}(c_{i-1}) = \sum_{\alpha \in H} F_i(\alpha)$ (defining $F_0(c_0)$ to be the element a). At the end of the interaction, the verifier also checks that $F_m(c_m) = F(c_1, \ldots, c_m)$, by querying F at the random location $(c_1, \ldots, c_m)$. This interactive proof is public-coin, and has m rounds, communication complexity $\mathrm{poly}(\log|\mathbb{F}|, m)$, and soundness error $\frac{md}{|\mathbb{F}|}$. The prover runs in time $\mathrm{poly}(\log|\mathbb{F}|, |H|^m)$ and space $\mathrm{poly}(\log|\mathbb{F}|, m, |H|)$ and the verifier runs in time $\mathrm{poly}(\log|\mathbb{F}|, m, d, |H|)$ and space $O(\log|\mathbb{F}| \cdot m)$.

The sumcheck protocol is *not* zero knowledge, because the prover reveals partial sums of F to the verifier. If we assume the existence of one-way functions, the protocol can be made computational zero knowledge by leveraging the fact that it is public-coin [GMR89; IY87; BGGHKMR88] (in fact, if we further assume the hardness of certain problems related to discrete logarithms then more efficient transformations are known [CD98]); moreover, there is strong evidence that assuming one-way functions is necessary [Ost91; OW93]. Even more, achieving statistical zero knowledge for sumcheck instances would cause unlikely complexity-theoretic collapses [For87; AH91].

Nevertheless, [BCFGRS16] have shown that, in the *Interactive PCP* model (see Section 4.3), a simple variant of the sumcheck protocol is *perfect zero knowledge*. The variant is as follows: the prover sends a proof oracle containing the evaluation of a random m-variate polynomial A of individual degree d, conditioned on summing to 0 on H^m; the verifier replies with a random element $\rho \in \mathbb{F}$; then the prover and verifier engage in a sumcheck protocol for the claim "$\sum_{\vec{\alpha} \in H^m} \rho F(\vec{\alpha}) + A(\vec{\alpha}) = a$", with the verifier accessing A via self-correction (after low-degree testing it). The proof oracle thus consists of $|\mathbb{F}|^m$ field elements, and the verifier accesses only $\mathrm{poly}(\log |\mathbb{F}|, m, d)$ of them.

The auxiliary polynomial A acts as a "masking polynomial", and yields the following zero knowledge guarantee: there exists a polynomial-time simulator algorithm that perfectly simulates the view of any malicious verifier, provided it can query F in as many locations as the *total* number of queries that the malicious verifier makes to either F or A.

5 Algebraic query complexity of polynomial summation

We have described in Section 2.2 an algebraic commitment scheme based on the sumcheck protocol and lower bounds on the algebraic query complexity of polynomial summation. The purpose of this section is to describe this construction in more detail, and then provide formal statements for the necessary lower bounds.

We begin with the case of committing to a single element $a \in \mathbb{F}$. The prover chooses a uniformly random string $B \in \mathbb{F}^N$ such that $\sum_{i=1}^{N} B_i = a$, for some $N \in \mathbb{N}$. Fixing some $d \in \mathbb{N}$, $G \subseteq \mathbb{F}$ and $k \in \mathbb{N}$ such that $|G| \leq d+1$ and $|G|^k = N$, the prover views B as a function from G^k to $\mathbb{F}$ (via an ordering on G^k) and sends the evaluation of a degree-d extension $\hat{B} \colon \mathbb{F}^k \to \mathbb{F}$ of B. The verifier tests that $\hat{B}$ is indeed (close to) a low-degree polynomial but (ideally) cannot learn any information about a without reading *all* of B (i.e., without making N queries). Subsequently, the prover can decommit to a by convincing the verifier that $\sum_{\vec{\beta} \in G^k} \hat{B}(\vec{\beta}) = a$ via the sumcheck protocol.

To show that the above is a commitment scheme, we must show both binding and hiding. Both properties depend on the choice of d. The binding property follows from the soundness of the sumcheck protocol, and we thus would like the degree d of $\hat{B}$ to be as small as possible. A natural choice would be $d = 1$ (so $|G| = 2$), which makes $\hat{B}$ the unique multilinear extension of B. However (as discussed in Section 2.2) this choice of parameters does not provide any hiding: it holds that $\sum_{\beta \in \{0,1\}^k} B(\beta) = \hat{B}(2^{-1}, \ldots, 2^{-1}) \cdot 2^k$ (as long as $\mathrm{char}(\mathbb{F}) \neq 2$). We therefore need to understand how the choice of d affects the number of queries to $\hat{B}$ required to compute a. This is precisely the setting of *algebraic query complexity*, which we discuss next.

The algebraic query complexity (defined in [AW09] to study 'algebrization') of a function f is the (worst-case) number of queries to some low-degree extension $\hat{B}$ of a string B required to compute $f(B)$. This quantity is bounded from above by the standard query complexity of f, but it may be the case (as above) that the low-degree extension confers additional information that helps in computing f with fewer queries. The usefulness of this information depends on parameters d and G of the low-degree extension. Our question amounts to understanding this dependence for the function $\textsc{Sum} \colon \mathbb{F}^N \to \mathbb{F}$ given by $\textsc{Sum}(B) := \sum_{i=1}^{N} B_i$. This has been studied before in [JKRS09]: if $G = \{0,1\}$ and $d = 2$ then the algebraic query complexity of $\textsc{Sum}$ is exactly N.

For our purposes, however, it is not enough to commit to a single field element. Rather, we need to commit to the evaluation of a polynomial $Q \colon \mathbb{F}^m \to \mathbb{F}$ of degree d_Q, which we do as follows. Let K be a subset of $\mathbb{F}$ of size $d_Q + 1$. The prover samples, for each $\vec{\alpha} \in K^m$, a random string $B^{\vec{\alpha}} \in \mathbb{F}^N$ such that $\textsc{Sum}(B^{\vec{\alpha}}) = Q(\vec{\alpha})$. The prover views these strings as a function $B \colon K^m \times G^k \to \mathbb{F}$, and takes a low-degree extension $\hat{B} \colon \mathbb{F}^m \times \mathbb{F}^k \to \mathbb{F}$. The polynomial $\hat{B}(\vec{X}, \vec{Y})$ has degree d_Q in $\vec{X}$ and d in $\vec{Y}$; this is a commitment to Q because $\sum_{\vec{\beta} \in G^k} \hat{B}(\vec{X}, \vec{\beta})$ is a degree-d_Q polynomial that agrees with Q on K^m, and therefore equals Q.

Once again we will decommit to $Q(\vec{\alpha})$ using the sumcheck protocol, and so for binding we need d to be small. For hiding, as in the single-element case, if d is too small then a few queries to $\hat{\mathcal{B}}$ can yield information about Q. Moreover, it could be the case that the verifier can leverage the fact that $\hat{\mathcal{B}}$ is a *joint* low-degree extension to learn some linear combination of evaluations of Q. We must exclude these possibilities in order to obtain our zero knowledge guarantees.

This question amounts to a generalization of algebraic query complexity where, given a list of strings $B_1, \ldots, B_M$, we determine how many queries we need to make to their *joint* low-degree extension $\hat{B}$ to determine any nontrivial linear combination $\sum_{i=1}^{M} c_i \cdot \textsc{Sum}(B_i)$. We will show that the 'generalized' algebraic query complexity of $\textsc{Sum}$ is exactly N provided $d \geq 2(|G| - 1)$ (which is also the case for the standard algebraic query complexity).

In the remainder of the section we state our results in a form equivalent to the above that is more useful to us. Given an arbitrary polynomial $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq d'}]$, we ask how many queries are required to determine any nontrivial linear combination of $\sum_{\vec{y} \in G^k} Z(\vec{\alpha}, \vec{y})$ for $\vec{\alpha} \in \mathbb{F}^m$. The following theorem is more general: it states that not only do we require many queries to determine *any* linear combination, but that the number of queries grows linearly with the number of independent combinations that we wish to learn.

Theorem 5.1 (algebraic query complexity of polynomial summation)**.** *Let $\mathbb{F}$ be a field, $m, k, d, d' \in \mathbb{N}$, and G, K, L be finite subsets of $\mathbb{F}$ such that $K \subseteq L$, $d' \geq |G| - 2$, and $|K| = d + 1$. If $S \subseteq \mathbb{F}^{m+k}$ is such that there exist matrices $C \in \mathbb{F}^{L^m \times \ell}$ and $D \in \mathbb{F}^{S \times \ell}$ such that for all $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq d'}]$ and all $i \in \{1, \ldots, \ell\}$*

$$\sum_{\vec{\alpha} \in L^m} C_{\vec{\alpha},i} \sum_{\vec{y} \in G^k} Z(\vec{\alpha}, \vec{y}) = \sum_{\vec{q} \in S} D_{\vec{q},i} Z(\vec{q}) \ ,$$

then $|S| \geq \operatorname{rank}(BC) \cdot (\min\{d' - |G| + 2, |G|\})^k$*, where* $B \in \mathbb{F}^{K^m \times L^m}$ *is such that column* $\vec{\alpha}$ *of* B *represents* $Z(\vec{\alpha})$ *in the basis* $(Z(\vec{\beta}))_{\vec{\beta} \in K^m}$*.*

We describe a special case of the above theorem that is necessary for our zero knowledge results, and then give an equivalent formulation in terms of random variables that we use in later sections. (Essentially, the linear structure of the problem implies that 'worst-case' statements are equivalent to 'average-case' statements.)

Corollary 5.2. *Let* $\mathbb{F}$ *be a finite field,* G *be a subset of* $\mathbb{F}$*, and* $d, d' \in \mathbb{N}$ *with* $d' \geq 2(|G| - 1)$*. If* $S \subseteq \mathbb{F}^{m+k}$ *is such that there exist* $(c_{\vec{\alpha}})_{\vec{\alpha} \in \mathbb{F}^m}$ *and* $(d_{\vec{\beta}})_{\vec{\beta} \in \mathbb{F}^{m+k}}$ *such that*

- *for all* $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq d'}]$ *it holds that* $\sum_{\vec{\alpha} \in \mathbb{F}^m} c_{\vec{\alpha}} \sum_{\vec{y} \in G^k} Z(\vec{\alpha}, \vec{y}) = \sum_{\vec{q} \in S} d_{\vec{q}} Z(\vec{q})$ *and*
- *there exists* $Z' \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq d'}]$ *such that* $\sum_{\vec{\alpha} \in \mathbb{F}^m} c_{\vec{\alpha}} \sum_{\vec{y} \in G^k} Z'(\vec{\alpha}, \vec{y}) \neq 0$*,*

then $|S| \geq |G|^k$*.*

Corollary 5.3 (equivalent statement of Corollary 5.2)**.** *Let* $\mathbb{F}$ *be a finite field,* G *be a subset of* $\mathbb{F}$*, and* $d, d' \in \mathbb{N}$ *with* $d' \geq 2(|G| - 1)$*. Let* Q *be a subset of* $\mathbb{F}^{m+k}$ *with* $|Q| < |G|^k$ *and let* Z *be uniformly random in* $\mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq d'}]$*. The ensembles* $\left(\sum_{\vec{y} \in G^k} Z(\vec{\alpha}, \vec{y})\right)_{\vec{\alpha} \in \mathbb{F}^m}$ *and* $\left(Z(\vec{q})\right)_{\vec{q} \in Q}$ *are independent.*

The proofs of these results, and derivations of corresponding upper bounds, are provided in Appendix A.

6 Zero knowledge sumcheck from algebraic query lower bounds

We leverage lower bounds on the algebraic query complexity of polynomial summation (Section 5) to obtain an analogue of the sumcheck protocol with a strong zero knowledge guarantee, which we use in the applications that we consider.

The sumcheck protocol [LFKN92] is an Interactive Proof for claims of the form $\sum_{\vec{x} \in H^m} F(\vec{x}) = a$, where H is a subset of a finite field $\mathbb{F}$, F is an m-variate polynomial over $\mathbb{F}$ of individual degree at most d, and a is an element of $\mathbb{F}$. The sumcheck protocol is *not* zero knowledge (conjecturally).

Prior work [BCFGRS16] obtains a sumcheck protocol, in the Interactive PCP model, with a certain zero knowledge guarantee. In that protocol, the prover first sends a proof oracle that consists of the evaluation of a random m-variate polynomial R of individual degree at most d; after that, the prover and the verifier run the (standard) sumcheck protocol on a new polynomial obtained from F and R. The purpose of R is to 'mask' the partial sums, which are the intermediate values sent by the prover during the sumcheck protocol.

The zero knowledge guarantee in [BCFGRS16] is the following: *any verifier that makes q queries to R learns at most q evaluations of F*. This guarantee suffices to obtain a zero knowledge protocol for $\#\mathbf{P}$ (the application in [BCFGRS16]) because the verifier can evaluate F efficiently at any point (as F is merely an arithmetization of a 3SAT formula).

We achieve a much stronger guarantee: *any verifier that makes polynomially-many queries to R learns at most a single evaluation of F* (that, moreover, lies within a chosen subset I^m of $\mathbb{F}^m$). Our applications require this guarantee because we use the sumcheck simulator as a sub-simulator in a larger protocol, where F is a randomized low-degree extension of some function that is hard to compute for the verifier. The randomization introduces bounded independence, which makes a small number of queries easy to simulate.

The main idea to achieve zero knowledge as above is the following. Rather than sending the masking polynomial R directly, the prover sends a (perfectly-hiding and statistically-binding) commitment to it in the form of a random $(m+k)$-variate polynomial Z. The 'real' mask is recovered by summing out k variables: $R(\vec{X}) := \sum_{\vec{\beta} \in G^k} Z(\vec{X}, \vec{\beta})$. Our lower bounds on the algebraic query complexity of polynomial summation (Section 5) imply that any q queries to Z, with $q < |G|^k$, yield *no information* about R. The prover, however, can elect to decommit to $R(\vec{c})$ for a single point $\vec{c} \in I^m$ chosen by the verifier. This is achieved using the zero knowledge sumcheck protocol of [BCFGRS16] as a subroutine: the prover sends $w := R(\vec{c})$ and then proves that $w = \sum_{\vec{\beta} \in G^k} Z(\vec{c}, \vec{\beta})$.

The protocol thus proceeds as follows. Given a security parameter $\lambda \in \mathbb{N}$, the prover sends the evaluations of two polynomials $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ and $A \in \mathbb{F}[Y_{1,\ldots,k}^{\leq 2\lambda}]$ as proof oracles. The verifier checks that both of these evaluations are close to low-degree, and uses self-correction to make for querying them. The prover sends two field elements z_1 and z_2, which are (allegedly) the summations of Z and A over $H^m \times G^k$ and G^k, respectively. The verifier replies with a random challenge $\rho \in \mathbb{F} \setminus \{0\}$. The prover and the verifier then engage in the standard (not zero knowledge) sumcheck protocol on the claim "$\sum_{\vec{\alpha} \in H^m} \rho F(\vec{\alpha}) + R(\vec{\alpha}) = \rho a + z_1$". This reduces the correctness of this claim to checking a claim of the form "$\rho F(\vec{c}) + R(\vec{c}) = b$" for some $\vec{c} \in I^m$ and $b \in \mathbb{F}$; the prover then decommits to $w := R(\vec{c})$ as above. In sum, the verifier deduces that, with high probability, the claim "$\rho F(\vec{c}) = b - w$" is true if and only if the original claim was.

If the verifier could evaluate F then the verifier could simply check the aforementioned claim and either accept or reject. We do not give the verifier access to F and, instead, we follow [Mei13] and phrase sumcheck as a *reduction* from a claim about a sum of a polynomial over a large product space to a claim about the evaluation of that polynomial at a single point. This view of the sumcheck protocol is useful later on when designing more complex protocols, which employ sumcheck as a subprotocol. The completeness and soundness definitions below are thus modified according to this viewpoint, where the verifier simply outputs the claim at the end.

Our protocol will be sound relative to a *promise variant* of the sumcheck protocol, which we now define.

Definition 6.1. *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 $\big((\mathbb{F}, m, d, H, a), F\big)$ such that:*
  - *$\mathbb{F}$ is a finite field, H is a subset of $\mathbb{F}$, a is an element of $\mathbb{F}$, and m, d are positive integers with $\frac{md}{|\mathbb{F}|} < \frac{1}{2}$;*
  - *F is a polynomial in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}]$ and sums to a on H^m.*
- *The* **sumcheck promise relation** *is the pair of relations $(\mathscr{R}_{\mathrm{SC}}^{\mathsf{yes}}, \mathscr{R}_{\mathrm{SC}}^{\mathsf{no}})$ where $\mathscr{R}_{\mathrm{SC}}^{\mathsf{yes}} := \mathscr{R}_{\mathrm{SC}}$ and $\mathscr{R}_{\mathrm{SC}}^{\mathsf{no}}$ are the pairs $\big((\mathbb{F}, m, d, H, a), F\big)$ such that $(\mathbb{F}, m, d, H, a)$ is as above and F is in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}]$ but does <u>not</u> sum to a on H^m.*

Remark 6.2. In the case where the verifier can easily determine that F is low-degree (e.g., F is given as an arithmetic circuit), a protocol for the promise relation can be used to check the plain relation. In our setting, the verifier cannot even access F, and so the promise is necessary.

The definition below captures our zero knowledge goal for the sumcheck promise relation, in the Interactive PCP model. The key aspect of this definition is that the simulator is only allowed to make a *single* query to the summand polynomial F; in contrast, the definition of [BCFGRS16] allows the simulator to make as many queries to F as the malicious verifier makes to the proof oracle. (Another aspect, motivated by the simulator's limitation, is that we now have to explicitly consider a bound b on a malicious verifier's queries.) This *strong* form of zero knowledge, achieved by severely restricting the simulator's access to F, is crucial for achieving the results in our paper.

Definition 6.3. *A b-**strong perfect zero knowledge Interactive PCP system for sumcheck** with soundness error ε is a pair of interactive algorithms (P, V) that satisfies the following properties.*

- COMPLETENESS. *For every $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{yes}}$, $V(\mathbb{F}, m, d, H, a)$ when interacting with $P^F(\mathbb{F}, m, d, H, a)$ outputs a claim of the form "$F(\vec{\gamma}) = a$" (with $\vec{\gamma} \in \mathbb{F}^m$ and $a \in \mathbb{F}$) that is true with probability* 1.
- SOUNDNESS. *For every $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{no}}$ and malicious prover $\tilde{P}$, $V(\mathbb{F}, m, d, H, a)$ when interacting with $\tilde{P}$ outputs a claim of the form "$F(\vec{\gamma}) = a$" (with $\vec{\gamma} \in \mathbb{F}^m$ and $a \in \mathbb{F}$) that is true with probability at most ε.*
- ZERO KNOWLEDGE. *There exists a straightline simulator S such that, for every instance-witness pair $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{yes}}$ and b-query malicious verifier $\tilde{V}$, the following two distributions are equal*

$$S^{\tilde{V}, F}(\mathbb{F}, m, d, H, a) \quad \textit{and} \quad \text{View}\ \langle P^F(\mathbb{F}, m, d, H, a), \tilde{V} \rangle \ .$$

  Moreover, the simulator S makes <u>only a single query</u> to F (at a location that possibly depends on $\tilde{V}$ and its random choices) and 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.

The main result of this section, stated below, is a construction that efficiently fulfills the definition above.

Theorem 6.4 (Strong PZK Sumcheck)**.** *For every positive integer λ with $\lambda \leq |\mathbb{F}|$, positive integer k, and subset I of $\mathbb{F}$, there exists a λ^k-strong perfect zero knowledge Interactive PCP system (P, V) for sumcheck with soundness error $\varepsilon = O(\frac{(m+k)\cdot(d+\lambda)}{|I|})$ and the following efficiency parameters.*

- Oracle round: *P sends an oracle proof string π, consisting of the evaluation tables of polynomials $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ and $A \in \mathbb{F}[Y_{1,\ldots,k}^{\leq 2\lambda}]$ drawn uniformly at random.*
- Interactive proof: *after the oracle round, P and V engage in an $(m + k + 1)$-round interactive proof; across the interaction, the verifier sends to the prover $m + k + 1$ field elements, while the prover sends to the verifier $O((m + k) \cdot d)$ field elements. (In particular, the interaction is public-coin.)*
- 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 time $|\mathbb{F}|^{O(m+k)}$ and space $\mathrm{poly}(\log|\mathbb{F}|, d^m, \lambda^k, |H|)$, and*
  - *V runs in time $\mathrm{poly}(\log|\mathbb{F}|, m, d, |H|, k, \lambda)$ and space $O((m + k)\log|\mathbb{F}|)$.*

In addition, there is a simulator S witnessing perfect zero knowledge for (P, V) such that S's single query to the summand polynomial F belongs to the set I^m, and S runs in time $\mathrm{poly}(|H|, m, d, k, \lambda, \log|\mathbb{F}|) \cdot \mathsf{q}_{\tilde{V}}^3$.

Remark 6.5. With two-way access to the random tape, the prover can be made to run in space $\mathrm{poly}(\log|\mathbb{F}|, d, m, \lambda, k, |H|)$.

We divide the proof in two steps. First (Section 6.1), we exhibit a protocol with the above properties in a hybrid model in which the prover and verifier have access to random low-degree polynomials. Second (Section 6.2), we use low-degree testing and self-correction to 'compile' this protocol into an Interactive PCP.

6.1 Step 1

We construct a public-coin Interactive Proof for sumcheck that achieves zero knowledge in a model where the prover and verifier have access to certain low-degree polynomials. In the soundness case, these may be arbitrary; in the zero knowledge case, these are random and depend only on the size parameters of the instance.

Construction 6.6. *Let G be any subset of $\mathbb{F}$ of size λ. In the Interactive Proof system $(P^\star_{\mathrm{IP}}, V^\star_{\mathrm{IP}})$:*

- *$P^\star_{\mathrm{IP}}$ and $V^\star_{\mathrm{IP}}$ receive a sumcheck instance $(\mathbb{F}, m, d, H, a)$ as common input;*
- *$P^\star_{\mathrm{IP}}$ and $V^\star_{\mathrm{IP}}$ receive polynomials $Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ and $A \in \mathbb{F}[Y^{\leq 2\lambda}_{1,\ldots,k}]$ as oracles;*
- *$P^\star_{\mathrm{IP}}$ additionally receives a summand polynomial $F \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}]$ as an oracle.*

The interaction between $P^\star_{\mathrm{IP}}$ and $V^\star_{\mathrm{IP}}$ proceeds as follows:

1. *$P^\star_{\mathrm{IP}}$ sends two elements in $\mathbb{F}$ to $V^\star_{\mathrm{IP}}$: $z_1 := \sum_{\vec{\alpha}\in H^m} \sum_{\vec{\beta}\in G^k} Z(\vec{\alpha}, \vec{\beta})$ and $z_2 := \sum_{\vec{\beta}\in G^k} A(\vec{\beta})$.*
2. *$V^\star_{\mathrm{IP}}$ draws a random element ρ_1 in $\mathbb{F} \setminus \{0\}$ and sends it to $P^\star_{\mathrm{IP}}$.*
3. *$P^\star_{\mathrm{IP}}$ and $V^\star_{\mathrm{IP}}$ run the sumcheck IP [LFKN92] on the statement "$\sum_{\vec{\alpha}\in H^m} Q(\vec{\alpha}) = \rho_1 a + z_1$" where*

$$Q(X_1, \ldots, X_m) := \rho_1 F(X_1, \ldots, X_m) + \sum_{\vec{\beta}\in G^k} Z(X_1, \ldots, X_m, \vec{\beta}) \ ,$$

   with $P^\star_{\mathrm{IP}}$ playing the role of the prover and $V^\star_{\mathrm{IP}}$ that of the verifier, and the following modification.

   For $i = 1, \ldots, m$, in the i-th round, $V^\star_{\mathrm{IP}}$ samples its random element c_i from the set I rather than from all of $\mathbb{F}$; if $P^\star_{\mathrm{IP}}$ ever receives $c_i \in \mathbb{F} \setminus I$, it immediately aborts. In particular, in the m-th round, $P^\star_{\mathrm{IP}}$ sends a polynomial $g_m(X_m) := \rho_1 F(c_1, \ldots, c_{m-1}, X_m) + \sum_{\vec{\beta}\in G^k} Z(c_1, \ldots, c_{m-1}, X_m, \vec{\beta})$ for some $c_1, \ldots, c_{m-1} \in I$.

4. *$V^\star_{\mathrm{IP}}$ sends $c_m \in I$ to $P^\star_{\mathrm{IP}}$.*
5. *$P^\star_{\mathrm{IP}}$ sends the element $w := \sum_{\vec{\beta}\in G^k} Z(\vec{c}, \vec{\beta})$ to $V^\star_{\mathrm{IP}}$, where $\vec{c} := (c_1, \ldots, c_m)$.*
6. *$V^\star_{\mathrm{IP}}$ draws a random element ρ_2 in $\mathbb{F} \setminus \{0\}$ and sends it to $P^\star_{\mathrm{IP}}$.*
7. *$P^\star_{\mathrm{IP}}$ and $V^\star_{\mathrm{IP}}$ engage in the sumcheck IP [LFKN92] on the claim "$\sum_{\vec{\beta}\in G^k} \rho_2 Z(\vec{c}, \vec{\beta}) + A(\vec{\beta}) = \rho_2 w + z_2$".*
8. *$V^\star_{\mathrm{IP}}$ outputs the claim "$F(\vec{c}) = \frac{g_m(c_m) - w}{\rho_1}$".*

We prove the following lemma about the construction above.

Lemma 6.7. *The IP system $(P^\star_{\mathrm{IP}}, V^\star_{\mathrm{IP}})$ satisfies the following properties.*

- COMPLETENESS. *For every instance-witness pair $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}^{\mathsf{yes}}_{\mathrm{SC}}$, polynomial $Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$, and polynomial $A \in \mathbb{F}[Y^{\leq 2\lambda}_{1,\ldots,k}]$, ${V^\star_{\mathrm{IP}}}^{Z,A}(\mathbb{F}, m, d, H, a)$ when interacting with ${P^\star_{\mathrm{IP}}}^{F,Z,A}(\mathbb{F}, m, d, H, a)$ outputs a claim of the form "$F(\vec{\gamma}) = a$" (with $\vec{\gamma} \in \mathbb{F}^m$ and $a \in \mathbb{F}$) that is true with probability* 1.
- SOUNDNESS. *For every instance-witness pair $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}^{\mathsf{no}}_{\mathrm{SC}}$, polynomial $Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$, polynomial $A \in \mathbb{F}[Y^{\leq 2\lambda}_{1,\ldots,k}]$, and malicious prover $\tilde{P}$, ${V^\star_{\mathrm{IP}}}^{Z,A}(\mathbb{F}, m, d, H, a)$ when interacting with $\tilde{P}$ outputs a claim of the form "$F(\vec{\gamma}) = a$" (with $\vec{\gamma} \in \mathbb{F}^m$ and $a \in \mathbb{F}$) that is true with probability at most $\frac{md}{|I|} + \frac{k \cdot 2\lambda + 2}{|\mathbb{F}| - 1}$.*
- ZERO KNOWLEDGE. *There exists a straightline simulator $S^\star_{\mathrm{IP}}$ such that, for every instance-witness pair $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}^{\mathsf{yes}}_{\mathrm{SC}}$ and λ^k-query malicious verifier $\tilde{V}$, the following two distributions are equal*

$$ {S^\star_{\mathrm{IP}}}^{\tilde{V},F}(\mathbb{F}, m, d, H, a) \quad \textit{and} \quad \mathrm{View}\ \langle {P^\star_{\mathrm{IP}}}^{F,Z,A}(\mathbb{F}, m, d, H, a), \tilde{V}^{Z,A} \rangle \ , $$

   where Z is uniformly random in $\mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ and A is uniformly random in $\mathbb{F}[Y^{\leq 2\lambda}_{1,\ldots,k}]$. Moreover:

– *$S^\star_{\mathrm{IP}}$ makes a single query to F at a point in I^m;*

– *$S^\star_{\mathrm{IP}}$ runs in time*

$$(m+k)((d+\lambda)\mathsf{q}_{\tilde{V}}|H| + (d+\lambda)^3\mathsf{q}_{\tilde{V}}^3)\cdot \mathrm{poly}(\log|\mathbb{F}|) = \mathrm{poly}(|H|, m, d, k, \lambda, \log|\mathbb{F}|)\cdot \mathsf{q}_{\tilde{V}}^3$$

where $\mathsf{q}_{\tilde{V}}$ is $\tilde{V}$'s query complexity;

– *$S^\star_{\mathrm{IP}}$'s behavior does not depend on a until after the simulated $\tilde{V}$ sends its first message.*

Proof. Completeness is clear from the protocol description and the completeness property of sumcheck. Soundness follows from the fact that, if $((\mathbb{F}, m, d, H, a), F) \in \mathscr{R}^{\mathsf{no}}_{\mathrm{SC}}$, we can argue as follows:

- For every polynomial $Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$, with probability $1 - \frac{1}{|\mathbb{F}|-1}$ over the choice of ρ_1, $\sum_{\vec{\alpha}\in H^m} Q(\vec{\alpha}) \neq \rho_1 a + z_1$, i.e., the sumcheck claim is false.
- Therefore, by the soundness guarantee of sumcheck, with probability at least $1 - md/|I|$, either the verifier rejects or $\rho_1 F(\vec{c}) + \sum_{\vec{\beta}\in G^k} Z(\vec{c},\vec{\beta}) \neq g_m(c_m)$.
- Finally, we distinguish two cases depending on $\tilde{P}$:
  – If $\tilde{P}$ sends $w \neq \sum_{\vec{\beta}\in G^k} Z(\vec{c},\vec{\beta})$, then $\sum_{\vec{\beta}\in G^k} \rho_2 Z(\vec{c},\vec{\beta}) + A(\vec{\beta}) \neq \rho_2 w + z_2$ with probability $1 - \frac{1}{|\mathbb{F}|-1}$ over the choice of ρ_2, for any choice of A. In this case, by the soundness guarantee of the sumcheck protocol the verifier rejects with probability at least $1 - \frac{k\cdot 2\lambda}{|\mathbb{F}|}$.
  – If $\tilde{P}$ sends $w = \sum_{\vec{\beta}\in G^k} Z(\vec{c},\vec{\beta})$, then $F(\vec{c}) \neq \frac{g_m(c_m)-w}{\rho_1}$ with probability 1.

Taking a union bound on the above cases yields the claimed soundness error.

To show the (perfect) zero knowledge guarantee, we need to construct a suitably-efficient straightline simulator that perfectly simulates the view of any malicious verifier $\tilde{V}$. We first construct an *inefficient* simulator $S^\star_{\mathrm{slow}}$, and prove that its output follows the desired distribution; afterwards, we explain how the simulator can be made efficient.

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

1. Draw $Z_{\mathrm{sim}} \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ and $A_{\mathrm{sim}} \in \mathbb{F}[Y^{\leq d}_{1,\ldots,k}]$ uniformly at random.
2. Begin simulating $\tilde{V}$. Its queries to Z and A are answered according to Z_{sim} and A_{sim} respectively.
3. Send $z^1_{\mathrm{sim}} := \sum_{\vec{\alpha}\in H^m}\sum_{\vec{\beta}\in G^k} Z(\vec{\alpha},\vec{\beta})$ and $z^2_{\mathrm{sim}} := \sum_{\vec{\beta}\in G^k} A(\vec{\beta})$ to $\tilde{V}$.
4. Receive $\tilde{\rho}_1$. Draw $Q_{\mathrm{sim}} \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}]$ uniformly at random conditioned on $\sum_{\vec{\alpha}\in H^m} Q_{\mathrm{sim}}(\vec{\alpha}) = \tilde{\rho}_1 a + z^1_{\mathrm{sim}}$, then engage in the sumcheck protocol on the claim "$\sum_{\vec{\alpha}\in H^m} Q_{\mathrm{sim}}(\vec{\alpha}) = \tilde{\rho}_1 a + z^1_{\mathrm{sim}}$". If in any round $\tilde{V}$ sends $c_i \notin I$ as a challenge, abort.
5. Let $\vec{c} \in I^m$ be the point chosen by $\tilde{V}$ in the sumcheck protocol above. Query $F(\vec{c})$, and draw $Z'_{\mathrm{sim}} \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ uniformly at random conditioned on
   - $\sum_{\vec{\alpha}\in H^m}\sum_{\vec{\beta}\in G^k} Z'_{\mathrm{sim}}(\vec{\alpha},\vec{\beta}) = z^1_{\mathrm{sim}}$,
   - $\sum_{\vec{\beta}\in G^k} Z'_{\mathrm{sim}}(\vec{c},\vec{\beta}) = w_{\mathrm{sim}}$, where $w_{\mathrm{sim}} := Q_{\mathrm{sim}}(\vec{c}) - \tilde{\rho}_1 F(\vec{c})$, and
   - $Z'_{\mathrm{sim}}(\vec{\gamma}) = Z_{\mathrm{sim}}(\vec{\gamma})$ for all previous queries $\vec{\gamma}$ to Z.

   From this point on, answer all queries to Z with Z'_{sim}.
6. Receive $\tilde{\rho}_2$ from $\tilde{V}$. Draw $Q'_{\mathrm{sim}} \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}]$ uniformly at random conditioned on
   - $\sum_{\vec{\beta}\in G^k} Q'_{\mathrm{sim}}(\vec{\beta}) = \tilde{\rho}_2 w_{\mathrm{sim}} + z^2_{\mathrm{sim}}$, and
   - $Q'_{\mathrm{sim}}(\vec{\gamma}) = \tilde{\rho}_2 Z'_{\mathrm{sim}}(\vec{c},\vec{\gamma}) + A_{\mathrm{sim}}(\vec{\gamma})$ for all previous queries $\vec{\gamma}$ to A.

   From this point on, answer all queries to $A(\vec{\gamma})$ with $Q'_{\mathrm{sim}}(\vec{\gamma}) - \tilde{\rho}_2 Z'_{\mathrm{sim}}(\vec{c},\vec{\gamma})$.
7. Engage in the sumcheck protocol on the claim "$\sum_{\vec{\beta}\in G^k} Q'_{\mathrm{sim}}(\vec{\beta}) = \tilde{\rho}_2 w_{\mathrm{sim}} + z^2_{\mathrm{sim}}$".
8. Output the view of the simulated $\tilde{V}$.

Let $Q(\vec{X}) := \tilde{\rho}_1 F(\vec{X}) + \sum_{\vec{\beta} \in G^k} Z(\vec{X}, \vec{\beta})$, and $Q'(\vec{Y}) := \tilde{\rho}_2 Z(\vec{c}, \vec{Y}) + A(\vec{Y})$. Observe that there exists a (deterministic) function $v(\cdot)$ such that

$$\text{View}\ \langle P^{\star}_{\text{IP}}{}^{F,Z,A}, \tilde{V}^{F,Z,A} \rangle = v(Q, Q', F, Z, Z, r) \quad \text{and} \quad S^{\star}_{\text{slow}}{}^{\tilde{V},F} = v(Q_{\text{sim}}, Q'_{\text{sim}}, F, Z_{\text{sim}}, Z'_{\text{sim}}, r) \ ,$$

where the random variable r is $\tilde{V}$'s private randomness. Indeed,

- $\tilde{V}$'s queries to Z up to Step 5 are answered by Z and Z_{sim} respectively, and after Step 5 by Z and Z'_{sim} respectively;
- $\tilde{V}$'s queries to $A(\cdot)$ are answered by $Q'(\cdot) - \tilde{\rho}_2 Z(\vec{c}, \cdot)$ and $Q'_{\text{sim}}(\cdot) - \tilde{\rho}_2 Z'_{\text{sim}}(\vec{c}, \cdot)$ respectively;
- the messages between $P^{\star}_{\text{IP}}$ and $\tilde{V}$ in the first sumcheck are a sumcheck on Q and Q_{sim} respectively;
- $w = Q(\vec{c}) - \tilde{\rho}_1 F(\vec{c})$ in one case and $w_{\text{sim}} = Q_{\text{sim}}(\vec{c}) - \tilde{\rho}_1 F(\vec{c})$ in the other case; and
- the messages between $P^{\star}_{\text{IP}}$ and $\tilde{V}$ in the second sumcheck are a sumcheck on Q' and Q'_{sim} respectively.

We now argue that the outputs of $v(\cdot)$ in the two cases are identically distributed:

$$v(Q, Q', F, Z, Z, r) \sim v(Q_{\text{sim}}, Q'_{\text{sim}}, F, Z_{\text{sim}}, Z'_{\text{sim}}, r) \ .$$

We do so via several 'hybrids', the first of which considers the distribution of Q. For every fixed $B \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}]$, consider the following probability value, where $Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ is uniformly random, and $U \subseteq \mathbb{F}^{m+k} \times \mathbb{F}$ is a set of query-answer pairs:

$$\begin{aligned} p(B|U) &:= \Pr_Z \left[Q(\vec{\alpha}) = B(\vec{\alpha}) \quad \forall\, \vec{\alpha} \in \mathbb{F}^m \;\middle|\; Z(\vec{\gamma}) = \omega \quad \forall\, (\vec{\gamma}, \omega) \in U \right] \\ &= \Pr_Z \left[\sum_{\vec{\beta} \in G^k} Z(\vec{\alpha}, \vec{\beta}) = B(\vec{\alpha}) - \tilde{\rho}_1 F(\vec{\alpha}) \quad \forall\, \vec{\alpha} \in \mathbb{F}^m \;\middle|\; Z(\vec{\gamma}) = \omega \quad \forall\, (\vec{\gamma}, \omega) \in U \right] \end{aligned}$$

The probability value $p(B|U)$ describes the distribution of Q in a real execution, when U is the set of query-answer pairs to Z made by $\tilde{V}$. By our lower bounds on the algebraic query complexity of polynomial summation (Corollary 5.3), whenever $|U| < \lambda^k$,

$$p(B|U) = \Pr_Z \left[\sum_{\vec{\beta} \in G^k} Z(\vec{\alpha}, \vec{\beta}) = B(\vec{\alpha}) - \tilde{\rho}_1 F(\vec{\alpha}) \quad \forall\, \vec{\alpha} \in \mathbb{F}^m \right] = 1/|\mathbb{F}[X^{\leq d}_{1,\ldots,m}]|.$$

since Z is uniformly random. Thus Q is uniformly random, and hence is identically distributed to Q_{sim}. Thus:

$$v(Q, Q', F, Z, Z, r) \sim v(Q_{\text{sim}}, Q', F, Z, Z, r) \ .$$

Next, $Q'(\vec{Y}) = \tilde{\rho}_2 Z(\vec{c}, \vec{Y}) + A(\vec{Y})$ is uniformly random such that $\sum_{\vec{\beta} \in G^k} Q'(\vec{\beta}) = \tilde{\rho}_2 w + z_2$, while Q'_{sim} is uniformly random such that $\sum_{\vec{\beta} \in G^k} Q'_{\text{sim}}(\vec{\beta}) = \tilde{\rho}_2 w_{\text{sim}} + z^2_{\text{sim}}$ and $Q'_{\text{sim}}(\vec{\gamma}_i) = \tilde{\rho}_2 Z'_{\text{sim}}(\vec{c}, \vec{\gamma}_i) + A_{\text{sim}}(\vec{\gamma}_i)$ for $\vec{\gamma}_1, \ldots \vec{\gamma}_k$ adversarially chosen. Since A_{sim} is itself uniformly random such that $\sum_{\vec{\beta} \in G^k} A_{\text{sim}}(\vec{\beta}) = 0$, this gives that $(Q', Z) \sim (Q'_{\text{sim}}, Z'_{\text{sim}})$, as Z and Z'_{sim} are both uniformly random summing to zero. Moreover, queries to Z'_{sim} are independent of Q_{sim} by Corollary 5.3. Thus,

$$v(Q_{\text{sim}}, Q', F, Z, Z, r) \sim v(Q_{\text{sim}}, Q'_{\text{sim}}, F, Z'_{\text{sim}}, Z'_{\text{sim}}, r) \ .$$

Finally we examine Z_{sim}, which is used only to answer queries to Z until Step 5. Note that Z'_{sim} is drawn identically to Z_{sim} except for the additional condition that $\sum_{\vec{\beta} \in G^k} Z'_{\text{sim}}(\vec{c}, \vec{\beta}) = v_{\text{sim}}$, and that $\tilde{V}$'s queries to Z remain consistent. By Corollary 5.3, the summation condition is independent of the answers to $\tilde{V}$'s queries to Z. So $\tilde{V}$'s queries to Z_{sim} are hence identically distributed to the same queries to Z'_{sim}, and thus

$$v(Q_{\text{sim}}, Q'_{\text{sim}}, F, Z'_{\text{sim}}, Z'_{\text{sim}}, r) \sim v(Q_{\text{sim}}, Q'_{\text{sim}}, F, Z_{\text{sim}}, Z'_{\text{sim}}, r) \ .$$

This concludes the argument for the correctness of the inefficient simulator $S^{\star}_{\text{slow}}$.

To complete the proof of zero knowledge, we note that $S^{\star}_{\text{slow}}$ can be transformed into an efficient simulator $S^{\star}_{\text{IP}}$ by using succinct constraint detection for the Reed–Muller code extended with partial sums [BCFGRS16]: more precisely, we can use the algorithm of Corollary 4.1 to answer both point and sum queries to Z, A, and Q, in a stateful way, maintaining corresponding tables $\mathsf{ans}_{Z_{\text{sim}}}$, $\mathsf{ans}_{A_{\text{sim}}}$, and $\mathsf{ans}_{Q_{\text{sim}}}$. □

6.2 Step 2

The Interactive Proof described and analyzed in Section 6.1 assumes that the prover and verifier have access to certain low-degree polynomials. We now use low-degree testing and self-correction to compile that Interactive Proof into an Interactive PCP, where the prover sends to the verifier evaluations of these polynomials as part of the proof oracle. This will conclude the proof of Theorem 6.4.

Proof of Theorem 6.4. Construct an IPCP system (P, V) for sumcheck as follows:

- The prover P, given input $(\mathbb{F}, m, d, H, a)$ and oracle access to F, samples polynomials $Z \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$, $A \in \mathbb{F}[Y_{1,\ldots,k}^{\leq 2\lambda}]$ uniformly at random, and sends their evaluations to the verifier V; then P simulates ${P_{\mathrm{IP}}^{\star}}^{F,Z,A}(\mathbb{F}, m, d, H, a)$.

- The verifier V, after receiving a proof string $\pi = (Z, A)$, simulates ${V_{\mathrm{IP}}^{\star}}^{F,\pi}(\mathbb{F}, m, d, H, a)$ up to V_{IP}'s two queries $\vec{\alpha}_1 \in \mathbb{F}^{m+k}, \vec{\alpha}_2 \in \mathbb{F}^k$ to Z, A respectively (which occur after the interaction), which V does not answer directly but instead answers as follows. First, V checks that Z is ϱ-close to the evaluation of a polynomial in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ 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 $Z(\vec{\alpha}_1)$ via self-correction with soundness error ϵ [RS96; AS03], and replies with this value. The procedures above are repeated to determine $A(\vec{\alpha}_2)$. Both procedures require $\mathrm{poly}(\log|\mathbb{F}|, m, d)$ 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 $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$. We now argue soundness. Consider an instance-witness pair $\big((\mathbb{F}, m, d, H, a), F\big) \in \mathscr{R}_{\mathrm{SC}}^{\mathsf{no}}$ and a malicious prover $\tilde{P}$, and denote by $\tilde{\pi} = (\tilde{Z}, \tilde{A})$ the proof string sent by $\tilde{P}$. We distinguish between the following two cases.

- *Case 1: $\tilde{Z}$ is ϱ-far from evaluations of polynomials in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ or $\tilde{A}$ is ϱ-far from evaluations of polynomials in $\mathbb{F}[Y_{1,\ldots,k}^{\leq 2\lambda}]$.*

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

- *Case 2: $\tilde{Z}$ is ϱ-close to evaluations of polynomials in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ and $\tilde{A}$ is ϱ-close to evaluations of polynomials in $\mathbb{F}[Y_{1,\ldots,k}^{\leq 2\lambda}]$.*

  In this case, let $\tilde{Z}'$ be the unique polynomial in $\mathbb{F}[X_{1,\ldots,m}^{\leq d}, Y_{1,\ldots,k}^{\leq 2\lambda}]$ whose evaluation is ϱ-close to $\tilde{Z}$; this polynomial exists because ϱ is less than the unique decoding radius (of the corresponding Reed–Muller code), which equals $\frac{1}{2}(1 - \frac{d}{|\mathbb{F}|})^m(1 - \frac{2\lambda}{|\mathbb{F}|})^k$, and is at least $\frac{1}{4}$ by the assumption that $\frac{md}{|\mathbb{F}|} + \frac{k \cdot 2\lambda}{|\mathbb{F}|} < \frac{1}{2}$. $\tilde{A}'$ is defined analogously. By the soundness of $(P_{\mathrm{IP}}, V_{\mathrm{IP}})$, the probability that $V_{\mathrm{IP}}^{F,\tilde{Z}',\tilde{A}'}$ accepts is at most $\frac{md}{|I|} + \frac{k \cdot 2\lambda + 2}{|\mathbb{F}|}$ (see Lemma 6.7). However V only has access to $\tilde{\pi}$, and uses self-correction on it to compute $\tilde{Z}', \tilde{A}'$ at the location $\vec{\alpha} \in \mathbb{F}^m$ required by V_{IP}; the probability that the returned values are not correct is at most 2ϵ. Hence, by a union bound, V accepts with probability at most $\frac{md}{|I|} + \frac{k \cdot 2\lambda + 2}{|\mathbb{F}|} + 2\epsilon$.

Overall, we deduce that V accepts with probability at most $\max\{\epsilon\,,\ \frac{md}{|I|} + \frac{k \cdot 2\lambda + 2}{|\mathbb{F}|} + 2\epsilon\} \leq 6\frac{(m+k)\cdot(d+\lambda)}{|I|}$. □

7 Zero knowledge for non-deterministic exponential time

In this section we use the zero knowledge sumcheck protocol developed in Section 6 (along with the [BCFGRS16] protocol) to derive a zero knowledge analogue of the [BFL91; BFLS91] protocol for $\mathbf{NEXP}$.[1]Recall that in this protocol, the prover first sends a low-degree extension of a $\mathbf{NEXP}$ witness, and then engages in the [LFKN92] sumcheck protocol on a polynomial related to the instance. To make this zero knowledge, the prover will first take a *randomized* low-degree extension R of the witness (which provides some bounded independence). The oracle contains a commitment to R: the prover draws a polynomial uniformly at random subject to the condition that 'summing out' a few of its variables yields R, and places its evaluation in the oracle.

The prover and verifier then engage in the zero knowledge sumcheck detailed in Section 6 on the [BFLS91] polynomial. This ensures that the verifier learns nothing through the interaction except for a single evaluation of the summand polynomial, which corresponds to learning a constant number of evaluations of the randomized witness. Bounded independence ensures that these evaluations do not leak any information. The prover provides these evaluations to the verifier, who will then check their correctness by engaging in an instance of the [BCFGRS16] sumcheck protocol for each evaluation. Note that here we are satisfied with the weaker guarantee provided by the [BCFGRS16] protocol because the simulator is able to simulate any polynomial number of queries to the commitment.

Following [BFLS91], the arithmetization encodes bit strings as elements in H^m for some H of size $\mathrm{poly}(|B|)$, rather than with $H = \{0,1\}$ as in [BFL91], for greater efficiency.

We start by defining the *oracle 3-satisfiability problem*, which is the $\mathbf{NEXP}$-complete problem used by [BFL91] to construct two-prover interactive proofs for $\mathbf{NEXP}$.

Definition 7.1 ($\mathscr{R}_{\mathrm{O3SAT}}$)**.** *The* **oracle 3-satisfiability relation***, denoted $\mathscr{R}_{\mathrm{O3SAT}}$, consists of all instance-witness pairs $(\mathtt{x}, \mathtt{w}) = \big((r, s, B), A\big)$, where r, s are positive integers, $B\colon \{0,1\}^{r+3s+3} \to \{0,1\}$ is a boolean formula, and $A\colon \{0,1\}^s \to \{0,1\}$ is a function, that satisfy the following condition:*

$$\forall\, z \in \{0,1\}^r,\ \forall\, b_1, b_2, b_3 \in \{0,1\}^s,\ B\big(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)\big) = 1\ .$$

Theorem 7.2 (PZK IPCP for $\mathbf{NEXP}$)**.** *For every query bound function $b(n)$, the $\mathbf{NEXP}$-complete relation $\mathscr{R}_{\mathrm{O3SAT}}$ has a (public coin and non-adaptive) Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail:*

$$\mathscr{R}_{\mathrm{O3SAT}} \in \mathbf{PZK\text{-}IPCP} \left[\begin{array}{rl} \textit{soundness error:} & 1/2 \\ \textit{round complexity:} & O(r+s+\log b) \\ \textit{proof length:} & \mathrm{poly}(2^{|B|}, b) \\ \textit{query complexity:} & \mathrm{poly}(|B| + \log b) \\ \textit{prover time:} & \mathrm{poly}(2^{|B|}, b) \\ \textit{verifier time:} & \mathrm{poly}(|B| + \log b) \\ \textit{verifier space:} & O(|B| + \log b) \\ \textit{simulator overhead:} & \mathrm{poly}(|B| + \log b) \cdot \mathsf{q}_V^3 \end{array}\right] .$$

Note that the prover running time given above assumes that the prover is given a witness as auxiliary input.

Proof. Let $\mathbb{F}$ be an extension field of $\mathbb{F}_2$. Let $\hat{B}\colon \mathbb{F}^m \to \mathbb{F}$ be the 'direct' arithmetization of the negation of B: rewrite B by using ANDs and NOTs; negate its output; replace each $\mathrm{AND}(a,b)$ with $a \cdot b$ and $\mathrm{NOT}(a)$ with $1 - a$. For every $\vec{x} \in \{0,1\}^{r+3s+3}$, $\hat{B}(\vec{x}) = 0$ if $B(\vec{x})$ is true, and $\hat{B}(\vec{x}) = 1$ if $B(\vec{x})$ is false. Note that $\hat{B}$ is computable in time $\mathrm{poly}(|B|)$ and has total degree $O(|B|)$.

Observe that $(r, s, B) \in \mathscr{R}_{\mathrm{O3SAT}}$ if and only if there exists a multilinear function $\hat{A}\colon \mathbb{F}^s \to \mathbb{F}$ that is boolean on $\{0,1\}^s$ such that $\hat{B}(\vec{z}, \vec{b}_1, \vec{b}_2, \vec{b}_3, \hat{A}(\vec{b}_1), \hat{A}(\vec{b}_2), \hat{A}(\vec{b}_3)) = 0$ for all $\vec{z} \in \{0,1\}^r$, $\vec{b}_1, \vec{b}_2, \vec{b}_3 \in \{0,1\}^s$. The requirement that $\hat{A}$ is boolean on $\{0,1\}^s$ can be encoded by 2^s constraints: $\hat{A}(\vec{b})(1 - \hat{A}(\vec{b})) = 0$ for every $\vec{b} \in \{0,1\}^s$.

[1]This section is written so that the proof can be understood independently of subsequent sections of the paper. Using the framework of sum-product circuits developed in Section 8, we can simplify this proof; see Appendix B.

These constraints can be expressed as follows:

$$\Big\{g_1(\vec{\alpha}) := \hat{B}(\vec{z}, \vec{b}_1, \vec{b}_2, \vec{b}_3, \hat{A}(\vec{b}_1), \hat{A}(\vec{b}_2), \hat{A}(\vec{b}_3)) = 0\Big\}_{\vec{z}\in\{0,1\}^r,\, \vec{b}_i\in\{0,1\}^s}$$
$$\Big\{g_2(\vec{\beta}) := \hat{A}(\vec{b})(1-\hat{A}(\vec{b})) = 0\Big\}_{\vec{b}\in\{0,1\}^s}$$

Let F be the polynomial over $\mathbb{F}$ given by

$$F(\vec{X}, \vec{Y}) := \sum_{\vec{\alpha}\in\{0,1\}^{r+3s}} \left(g_1(\vec{\alpha})\vec{X}^{\vec{\alpha}} + g_2(\vec{\alpha}_{[s]})\vec{Y}^{\vec{\alpha}}\right) \ ,$$

where $\vec{X}^{\vec{\alpha}} := X_1^{\alpha_1}\cdots X_\ell^{\alpha_\ell}$ for $\vec{\alpha}\in\{0,1\}^\ell$, and $\vec{\alpha}_{[s]}$ are the first s coordinates in $\vec{\alpha}$.

Note that F is the zero polynomial if and only if all the above equations hold. Since F is a polynomial of total degree $r+3s$, if F is not the zero polynomial then it is zero on at most an $\frac{r+3s}{|\mathbb{F}|}$ fraction of points in $\mathbb{F}^{2(r+3s)}$.

For $\alpha_i\in\{0,1\}$ it holds that $X_i^{\alpha_i} = 1+(X_i-1)\alpha_i$, so we can also write

$$F(\vec{X}, \vec{Y}) = \sum_{\vec{\alpha}\in\{0,1\}^{r+3s}} \left(g_1(\vec{\alpha})\cdot\prod_{i=1}^{r+3s}(1+(X_i-1)\alpha_i) + g_2(\vec{\alpha}_{[s]})\cdot\prod_{i=1}^{r+3s}(1+(Y_i-1)\alpha_i)\right) =: \sum_{\vec{\alpha}\in\{0,1\}^{r+3s}} f(\vec{X},\vec{Y},\vec{\alpha}) \ .$$

Let H be a subfield of $\mathbb{F}$; define $m_1 := r/\log|H|$ and $m_2 := s/\log|H|$ (assuming without loss of generality that both are integers). For $i\in\{1,2\}$, let $\gamma_i\colon H^{m_i}\to\{0,1\}^{m_i\log|H|}$ be the lexicographic order on H^{m_i}. The low-degree extension $\hat{\gamma}_i$ of γ_i is computable by an arithmetic circuit constructible in time $\mathrm{poly}(|H|, m_i, \log|\mathbb{F}|)$ [GKR15, Claim 4.2]. Let $\gamma\colon H^{m_1+3m_2}\to\{0,1\}^{r+3s}$ be such that $\gamma(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3) = (\gamma_1(\vec{\alpha}),\gamma_2(\vec{\beta}_1),\gamma_2(\vec{\beta}_2),\gamma_2(\vec{\beta}_3))$ for all $\vec{\alpha}\in H^{m_1}$, $\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3\in H^{m_2}$; let $\hat{\gamma}\colon\mathbb{F}^{m_1+3m_2}\to\mathbb{F}^{r+3s}$ be its low-degree extension.

We can use the above notation to write F equivalently as

$$F(\vec{X},\vec{Y}) = \sum_{\substack{\vec{\alpha}\in H^{m_1}\\ \vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3\in H^{m_2}}} g_1(\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3))\prod_{i=1}^{r+3s}(1+(X_i-1)\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3)_i)$$
$$+ g_2(\hat{\gamma}_2(\vec{\beta}_1))\prod_{i=1}^{r+3s}(1+(Y_i-1)\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3)_i) \ .$$

We are now ready to specify the protocol. Let $k := \lceil \log b/\log|H|\rceil$.

1. The prover draws a polynomial Z uniformly at random from $\mathbb{F}[X_{1,\ldots,m_2}^{\le|H|+2}, Y_{1,\ldots,k}^{\le 2|H|}]$, subject to the condition that $\sum_{\vec{\beta}\in G^k} Z(\vec{\alpha},\vec{\beta}) = A(\gamma_2(\vec{\alpha}))$ for all $\vec{\alpha}\in H^{m_2}$. It then generates an oracle π_0 for the $|H|^k$-strong zero knowledge sumcheck protocol (Section 6) on input $(\mathbb{F}, m_1+3m_2, \deg(f), H, 0)$ and oracles π_1,π_2,π_3 for the [BCFGRS16] zero knowledge sumcheck protocol on input $(\mathbb{F}, k, 2|H|, H, \cdot)$. (Recall that in both zero knowledge sumchecks, the oracle message does not depend on the claim itself.) The prover sends an oracle which is the concatenation of the evaluation of Z with $(\pi_0,\pi_1,\pi_2,\pi_3)$.

2. The verifier chooses $\vec{x},\vec{y}\in\mathbb{F}^{r+3s}$ uniformly at random and sends them to the prover. The prover and verifier engage in the zero knowledge sumcheck protocol of Section 6 on the claim "$F(\vec{x},\vec{y}) = 0$" with $I = \mathbb{F}\setminus H$ using π_1 as the oracle message. This reduces the claim to checking that $f(\vec{x},\vec{y},\vec{c},\vec{c}'_1,\vec{c}'_2,\vec{c}'_3) = a$ for uniformly random $\vec{c}\in(\mathbb{F}\setminus H)^{m_1}$, $\vec{c}'_1,\vec{c}'_2,\vec{c}'_3\in(\mathbb{F}\setminus H)^{m_2}$ and some $a\in\mathbb{F}$ provided by the prover.

3. The prover provides $h_i := A(\gamma_2(\vec{c}'_i))$ for each $i\in\{1,2,3\}$. The verifier substitutes these values into the expression for f to check the above claims, and rejects if they do not hold.

4. The prover and verifier engage in the zero knowledge sumcheck protocol of [BCFGRS16] on the claims "$\sum_{\vec{\beta}\in H^k} Z(\vec{\alpha},\vec{\beta}) = h_i$" for each $i\in\{1,2,3\}$, using π_i as the oracle message.

5. The verifier checks that Z is low-degree (with proximity parameter $\varrho := \frac{1}{8}$ and soundness error $\varepsilon := \frac{1}{|\mathbb{F}|}$), and uses self-correction (with soundness error ε) to query it at the points required by the [BCFGRS16] protocol above.

Completeness. If $((r, s, B), A) \in \mathscr{R}_{\mathrm{O3SAT}}$ then $F(\vec{X}, \vec{Y})$ is the zero polynomial; hence $F(\vec{x}, \vec{y}) = 0$ for all $\vec{x}, \vec{y} \in \mathbb{F}^{r+3s}$. Completeness follows from the completeness of the zero knowledge sumcheck protocols.

Soundness. Suppose that $(r, s, B) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ and let $(\tilde{Z}, \tilde{\pi}_0, \tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3)$ be the oracle message. If $\tilde{Z}$ is ϱ-far from an evaluation of a polynomial in $\mathbb{F}[X_{1,\ldots,m_2}^{\leq |H|+2}, Y_{1,\ldots,k}^{\leq \lambda}]$ then the verifier rejects with probability at least $1 - \varepsilon$. Otherwise, there exists a unique polynomial $Z \in \mathbb{F}[X_{1,\ldots,m_2}^{\leq |H|+2}, Y_{1,\ldots,k}^{\leq \lambda}]$ whose evaluation is ϱ-close to $\tilde{Z}$. Let $\tilde{A} := \sum_{\vec{\beta} \in H^k} Z(\vec{X}, \vec{\beta})$, which we think of as playing the role of $\hat{A}(\gamma_2(\cdot))$ in F.

If $(r, s, B) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ then there is no choice of $\hat{A}$ such that $F(\vec{X}, \vec{Y})$ is the zero polynomial. Thus, $F(\vec{x}, \vec{y}) = 0$ with probability at most $(r + 3s)/|\mathbb{F}|$ over the choice of $\vec{x}, \vec{y}$. By the soundness of the zero knowledge sumcheck protocol (Theorem 6.4), the verifier outputs a false claim "$f(\vec{x}, \vec{y}, \vec{\alpha}) = a$" with probability at least $1 - O((m_1 + m_2 + k)|H|)/(|\mathbb{F}| - |H|))$. If substituting h_i for $\hat{A}(\gamma_2(\vec{c}_i'))$ in f does not yield a, then the verifier rejects. Otherwise, it must be the case that for at least one $i \in \{1, 2, 3\}$, $\tilde{A}(\vec{c}_i') \neq h_i$. By the soundness of the [BCFGRS16] sumcheck protocol, the verifier rejects with probability at least $1 - O(\frac{k|H|}{|\mathbb{F}|})$. Taking a union bound, the verifier rejects with probability at least $1 - O((m_1 + m_2 + k)|H|/|\mathbb{F}|) = 1 - O((r + s + \log b)|H|/|\mathbb{F}|)$.

Zero knowledge. Perfect zero knowledge for this protocol is witnessed by the following simulator.

1. Draw a uniformly random polynomial $Z_{\mathrm{sim}} \in \mathbb{F}[X_{1,\ldots,m_2}^{\leq |H|+2}, Y_{1,\ldots,k}^{\leq 2|H|}]$.
2. Run the $|H|^k$-strong ZK sumcheck simulator on input $(\mathbb{F}, m_1 + 3m_2, \deg(f), H, 0)$, and use it to answer queries to π_0 throughout. In parallel, run three copies of the simulator for the [BCFGRS16] sumcheck on input $(\mathbb{F}, k, 2|H|, H, \cdot)$, and use them to answer queries to π_1, π_2, π_3 respectively. Recall that the behavior of each simulator does not depend on the claim being proven until after the first simulated message, so we can choose these later.
3. Receive $\vec{x}, \vec{y} \in \mathbb{F}^{r+3s}$ from $\tilde{V}$.
4. Simulate the strong ZK sumcheck protocol on the claim "$F(\vec{x}, \vec{y}) = 0$". The subsimulator will query f at a single location $\vec{c} \in (\mathbb{F} - H)^{r+3s}$. Reply with the value $f(\vec{x}, \vec{y}, \vec{c})$, for $\vec{c} = (\vec{c}_0, \vec{c}_1, \vec{c}_2, \vec{c}_3) \in (\mathbb{F} \setminus H)^{r+3s}$. To compute this requires values $\hat{A}(\hat{\gamma}(\vec{c}_i))$ for $i \in \{1, 2, 3\}$; we substitute each of these with $h_{\mathrm{sim}}^i \in \mathbb{F}$ drawn uniformly at random (except: if $\vec{c}_i = \vec{c}_j$ for $i \neq j$ then fix $h_{\mathrm{sim}}^i = h_{\mathrm{sim}}^j$).
5. For $i \in \{1, 2, 3\}$, simulate the [BCFGRS16] sumcheck protocol on the claim "$\sum_{\vec{\beta} \in H^k} Z(\vec{\alpha}, \vec{\beta}) = h_{\mathrm{sim}}^i$". Whenever the subsimulator queries Z, answer using Z_{sim}.

The verifier's view consists of its interaction with P during the four sumchecks, and its queries to the oracle. The Section 6 zero knowledge sumcheck subsimulator guarantees that the queries to π_0 and the first sumcheck are perfectly simulated given a single query to f at the point $\vec{c} \in (\mathbb{F} \setminus H)^{r+3s}$ chosen by $\tilde{V}$. Since $\hat{A}'(\vec{X}) = \sum_{\vec{\beta} \in H^k} Z(\vec{X}, \vec{\beta}) \in \mathbb{F}[X_{1,\ldots,m}^{\leq |H|+2}]$, the evaluation of $\hat{A}$ at any 3 points outside of H^m does not determine its value at any point in H^m. In particular, this means that the values h_i sent by the prover in the original protocol are independently uniformly random in $\mathbb{F}$ (except if $\vec{c}_i = \vec{c}_j$ for $i \neq j$ as above). Thus the h_{sim}^i are identically distributed to the h_i, and therefore both the prover message and the simulator's query are perfectly simulated.

The [BCFGRS16] sumcheck simulator ensures that the view of the verifier in the rest of the sumchecks is perfectly simulated given $q_{\tilde{V}}$ queries to Z, where $q_{\tilde{V}}$ is the number of queries the verifier makes across all π_i, $i \in \{1, 2, 3\}$. Hence the number of 'queries' the simulator makes to Z_{sim} is strictly less than b (because $\tilde{V}$ is b-query). By Corollary 5.3, any set of strictly less than b queries to Z is independent of $\hat{A}'$, and so the answers are identically distributed to the answers to those queries if they were made to a uniformly random polynomial, which is the distribution of Z_{sim}.

Clearly drawing a uniformly random polynomial in $Z_{\mathrm{sim}} \in \mathbb{F}[X_{1,\ldots,m_2}^{\leq |H|+2}, Y_{1,\ldots,k}^{\leq 2|H|}]$ is not something we can do in polynomial time. However, we can instead use the algorithm of Corollary 4.1 to draw Z (a simple modification allows us to handle different degrees in $\vec{X}, \vec{Y}$, or we could simply set the degree bound for both to be $2|H|$; the proof still goes through). The running time of the simulator is then $\mathrm{poly}(m_1, m_2, k, |H|, \log |\mathbb{F}|)$.

It remains to choose $\mathbb{F}$ and H. We set $|H| = \mathrm{poly}(r + s + \log b)$ and $|\mathbb{F}| = \mathrm{poly}(|H|)$ large enough that the soundness error is $o(1)$. The running time of the verifier is then $\mathrm{poly}(|B|, \log b)$, as is the running time of the simulator. The proof length is $\mathbb{F}^{O(m_1+m_2+k)} = 2^{O(r+s)} \cdot \mathrm{poly}(b)$. □

8 Delegating sum-product computations

We define *sum-product circuits*, a type of computation involving alternations of (i) summing polynomials over hypercubes, and (ii) combining polynomials via low-degree arithmetic circuits. Computing the output of a sum-product circuit is (conjecturally) hard (indeed, we will show that it is **PSPACE**-complete), but we show how to efficiently delegate such computations via an Interactive Proof.

We proceed in three steps. First, we provide intuition for why it is natural to consider sum-product alternations (Section 8.1). Then, we define sum-product *formulas*, which are a special case (in a way that is analogous to how boolean formulas specialize boolean circuits) and show how to delegate their evaluation (Section 8.2). Finally, we define sum-product *circuits* and show how to delegate their evaluation (Section 8.3).

In later sections, we additionally achieve zero knowledge via an Interactive PCP (Section 9), and explain how to 'program' sum-product circuits so that: their evaluation captures **PSPACE** (Section 10) or, more generally, low-depth circuit computations (Section 11); and their satisfaction captures **NEXP** (Section 7).

8.1 Intuition for definition

We provide intuition for why it is natural to consider sum-product alternations. Let $\mathbb{F}$ be a finite field, H a subset of $\mathbb{F}$, and m a positive integer.[2] The sumcheck protocol (Section 4.5) supports checking claims of the form "$a = \sum_{\vec{\beta} \in H^m} P(\vec{\beta})$" for a given field element $a \in \mathbb{F}$ and low-degree m-variate polynomial P over $\mathbb{F}$, if the verifier can efficiently evaluate P at any point (e.g., the verifier has a small arithmetic circuit for P, or the verifier has oracle access to P, or others).

Can the verifier still check the claim even if P is an expression involving other polynomials?

Suppose that $P(\vec{X})$ (allegedly) equals $C(\vec{X}, P_1(\vec{X}), \ldots, P_t(\vec{X}))$ for some low-degree t-variate 'combiner' polynomial C and low-degree m-variate polynomials $P_1, \ldots, P_t$, and suppose that the verifier has small arithmetic circuits for all these polynomials. In this case the verifier can still efficiently evaluate P at any given point, and the sumcheck protocol directly applies. However, now suppose instead that each polynomial $P_i(\vec{X})$ *itself* (allegedly) equals $\sum_{\vec{\gamma} \in H^m} C(\vec{X}, P_{i,1}(\vec{X}, \vec{\gamma}), \ldots, P_{i,t}(\vec{X}, \vec{\gamma}))$ for some low-degree $2m$-variate polynomials $P_{i,1}, \ldots, P_{i,t}$. Now the sumcheck protocol *does not* directly apply, due to the *alternation* of sums and products. What to do?

Sum-product expressions, and protocols for them. More generally (and informally), we call $P \colon \mathbb{F}^m \to \mathbb{F}$ an m-variate *sum-product expression* if (i) P is a low-degree (individual degree less than $|H|$) arithmetic circuit, or (ii) $P(\vec{X})$ equals $\sum_{\vec{\beta} \in H^m} C(\vec{X}, \vec{\beta}, P_1(\vec{X}, \vec{\beta}), \ldots, P_t(\vec{X}, \vec{\beta}))$ where C is a low-degree 'combiner' arithmetic circuit and $P_1, \ldots, P_t$ are $2m$-variate sum-product expressions.

By building on ideas of [Sha92; She92; GKR15], we can *still* use the sumcheck protocol, now as a subroutine of a larger Interactive Proof, to verify claims of the form "$a = P(\vec{\omega})$" for a given $a \in \mathbb{F}$, sum-product expression P, and $\vec{\omega} \in \mathbb{F}^m$, as we now sketch — and thereby handle sum-product alternations.

If P is an arithmetic circuit, then the verifier can check the claim directly by evaluating P at $\vec{\omega}$. Otherwise, proceed as follows. Define $\hat{P}$ to be the low-degree extension of P (see Section 4.1):

$$\hat{P}(X) = \sum_{\vec{\alpha} \in H^m} I_{H^n}(\vec{X}, \vec{\alpha}) \sum_{\vec{\beta} \in H^m} C(\vec{\alpha}, \vec{\beta}, P_1(\vec{\alpha}, \vec{\beta}), \ldots, P_t(\vec{\alpha}, \vec{\beta})) \ .$$

Recall that $I_{H^m}(\vec{X}, \vec{Y})$ is the unique m-variate polynomial, of degree less than $|H|$, such that, for all $(\vec{\alpha}, \vec{\beta}) \in H^m \times H^m$, $I_{H^m}(\vec{\alpha}, \vec{\beta})$ equals 1 when $\vec{\alpha} = \vec{\beta}$ and equals 0 otherwise.

The prover and verifier run the sumcheck protocol on the claim "$a = \hat{P}(\vec{\omega})$" and obtain a new claim

$$\text{"}a' = I_{H^n}(\vec{\omega}, \vec{r}_2) \cdot C(\vec{r}_1, \vec{r}_2, P_1(\vec{r}_1, \vec{r}_2), \ldots, P_t(\vec{r}_1, \vec{r}_2))\text{"}$$

for some $a' \in \mathbb{F}$ derived from the prover's messages and $\vec{r}_1, \vec{r}_2 \in \mathbb{F}^m$ drawn uniformly at random by the verifier. The prover then sends $h_1, \ldots, h_t$ and the verifier checks that $a' = I_{H^n}(\vec{\omega}, \vec{r}_2) \cdot C(h_1, \ldots, h_t)$. (Note that this expression

[2] Throughout, we assume that $|\mathbb{F}|$ is at least a constant fraction larger than $|H|$. In fact, we will typically take $|\mathbb{F}|$ to be larger than this (e.g., at least polynomial in $|H|$) to achieve good soundness.

involves only low-degree polynomials.) The verifier then recursively checks, for $i = 1, \ldots, t$, that "$a_i = P_i(\vec{r}_1, \vec{r}_2)$", relying on the fact that each P_i is itself a sum-product expression.

The reason for taking the low-degree extension $\hat{P}$ of P is to prevent a degree blowup for intermediate claims, and is also used in the GKR protocol [GKR15] as well as Shen's protocol [She92] (known as *degree reduction* there). In particular, depending on the form of the combiner C, the degree of $P(\vec{X})$ can be somewhat larger than that of the P_i subexpressions. Even a factor 2 increase in the degree would, after k rounds, lead to a factor 2^k increase overall, which for modest k would make the communication complexity of the sumcheck protocol superpolynomial. The degree reduction step ensures that the degrees of the intermediate claims do not increase.

Towards sum-product formulas. The above informal discussion motivates the formulation of tree-like computations that combine values of previous hypercube sums by way of functions of bounded degree — we call these *sum-product formulas* (in analogy to boolean formulas that are also tree-like computations). Our definition also features crucial degrees of freedom, which make 'programming' these formulas more efficient, that we now discuss.

First, we allow each internal vertex v in the tree to be labeled with a potentially different combiner arithmetic circuit C_v of small (total) degree. An input $\mathtt{x}$ to the sum-product formula then consists of labeling each leaf vertex v with a polynomial $\mathtt{x}_v$, potentially represented as an arithmetic circuit, of small (individual) degree, and the edges in the tree determine how to 'evaluate' a vertex, as follows. The *value* of a vertex v on input $\mathtt{x}$ equals the circuit $\mathtt{x}_v$ if v is a leaf vertex, or equals $\sum_{\vec{\beta} \in H^m} C_v\big(u_1[\mathtt{x}](\vec{X}, \vec{\beta}) \ldots, u_t[\mathtt{x}](\vec{X}, \vec{\beta})\big)$ if v is an internal vertex, where $u_1, \ldots, u_t$ are the children of v. The value of the formula on $\mathtt{x}$ is the value of the root on $\mathtt{x}$.

Second, we allow flexible 'arity' in the sums: each edge e is labeled with finite sets of positive integers ρ_e and σ_e that determine which 'free variables' and 'summation variables' are passed on to the child corresponding to e. In other words, now the recursion looks like $\sum_{\vec{\beta} \in H^{m_v}} C_v\big(u_1[\mathtt{x}](\vec{X}|_{\rho_{e_1}}, \vec{\beta}|_{\sigma_{e_1}}), \ldots, u_t[\mathtt{x}](\vec{X}|_{\rho_{e_t}}, \vec{\beta}|_{\sigma_{e_t}})\big)$ where $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ and $m_v := \max(\sigma_{e_1} \cup \cdots \cup \sigma_{e_{\mathsf{out}_T(v)}})$.

In Section 8.2 we provide the formal definition of sum-product formulas, and also describe how to outsource computations about them. We provide this only as a simpler stepping stone towards the next definition.

Sum-product circuits: re-using sub-computations. A boolean formula is limited in that it cannot re-use sub-computations; a sum-product formula is similarly limited. Thus, in analogy to boolean circuits, we consider *sum-product circuits*, in which sub-computations can be re-used according to an underlying directed acyclic graph (in fact, we will need a multi-graph), rather than a tree. Of course, one can always reduce sum-product circuits to sum-product formulas by 'opening up' the graph into a tree — but in the worst case this incurs an exponential blowup in the resulting tree. We use a standard trick to modify the protocol for sum-product formulas so to support merging multiple sub-claim computations at a vertex into one claim (regardless of the in-degree of the vertex), which avoids this explosion. This is also a necessary step in GKR's protocol [GKR15] (though we implement it differently for compatibility with our zero-knowledge protocols).

In Section 8.3 we provide a formal definition of sum-product circuits, and then describe how to extend the ideas discussed so far to also support outsourcing computations about sum-product circuits. In the rest of the paper we only use (and must use) sum-product circuits, as sum-product formulas are not expressive enough for our purposes.

8.2 Sum-product formulas

The purpose of this section is to (i) introduce *sum-product formulas*, and (ii) give proof systems for two computational problems about these, *evaluation* and *satisfaction*.

8.2.1 Formal definition

As with a boolean formula, the 'topology' of a a sum-product formula is a *tree*. A (rooted) tree $T = (V, E)$ is an acyclic connected graph in which edges are directed away from a distinguished vertex, known as the root of T and denoted r_T; the sinks of the graph are known as the leaves of T while all other vertices are known as internal vertices. The *depth* of a vertex v, denoted $\mathsf{depth}_T(v)$, is the number of edges on the path from r_T to v (thus r_T has depth 0). The depth of T, denoted $\mathsf{depth}(T)$, is the maximum depth of any vertex v in V. The width of T, denoted $\mathsf{width}(T)$, is the maximum number of vertices at any depth: $\max_{i=1}^{\mathsf{depth}(T)} |\{v \in V : \mathsf{depth}_T(v) = i\}|$. The *out-degree* of a vertex v is denoted $\mathsf{out}_T(v)$ and equals the number of children of v; the *in-degree* is 1 for all vertices except the root.

As outlined in Section 8.1, we eventually consider trees in which each internal vertex specifies a function that is recursively defined in terms of its children's functions. The number of inputs to these functions varies from vertex to vertex, and we specify the *arity* of these functions via certain edge labels. Namely, each edge $e = (u, v)$ is labeled by two "projections" ρ_e and σ_e that, respectively, specify which free variables of u (the $\vec{X}$ part in Equation 5 below) and summation variables of u (the $\vec{\beta}$ part in Equation 5 below) are passed on to v. In order for these projections to yield a well-defined notion of arity, they must satisfy certain consistency properties, and this motivates the following definition.

Definition 8.1. *A tuple $T = (V, E, \rho, \sigma)$ is an* **ari-tree** *if (V, E) is a tree and both ρ and σ label every edge e in E with finite sets of positive integers ρ_e and σ_e that satisfy the following property. For every vertex v in V, there exists a (unique) non-negative integer $\mathsf{arity}(v)$ such that: (1) if v is the root then $\mathsf{arity}(v) = 0$, otherwise $\mathsf{arity}(v) = |\rho_e| + |\sigma_e|$ where e is v's (unique) incoming edge; (2) $\rho_{e_1}, \ldots, \rho_{e_t} \subseteq \{1, \ldots, \mathsf{arity}(v)\}$, where $e_1, \ldots, e_t$ are v's outgoing edges.*

For convenience, we denote by $\mathsf{arity}(T)$ the maximum of $\mathsf{arity}(v)$ across all vertices v in the vertex set V of T. Moreover, for every vertex v, we define $m_v := \max(\sigma_{e_1} \cup \cdots \cup \sigma_{e_t})$ so that $\sigma_{e_1}, \ldots, \sigma_{e_t} \subseteq \{1, \ldots, m_v\}$.

We are now ready to define a sum-product formula $\mathcal{F}$, an input $\mathtt{x}$ for $\mathcal{F}$, and how to evaluate $\mathcal{F}$ on $\mathtt{x}$.

Definition 8.2. *A* **sum-product formula** *is a tuple $\mathcal{F} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, T, C)$ where: $\mathbb{F}$ is a finite field, H is a subset of $\mathbb{F}$ (represented as a list of field elements), $\delta_{\mathsf{in}}, \delta_{\mathsf{lf}}$ are positive integers (represented in unary) with $\delta_{\mathsf{lf}} \geq |H|$, $T = (V, E, \rho, \sigma)$ is an ari-tree, and C labels each internal vertex v of T with an arithmetic circuit $C_v(\vec{X}, \vec{Y}, \vec{Z}) \colon \mathbb{F}^{\mathsf{arity}(v)} \times \mathbb{F}^{m_v} \times \mathbb{F}^{\mathsf{out}_T(v)} \to \mathbb{F}$ of <u>total</u> degree at most δ_{in}. An* **input** $\mathtt{x}$ *for $\mathcal{F}$ labels each leaf vertex v of T with a polynomial $\mathtt{x}_v \colon \mathbb{F}^{\mathsf{arity}(v)} \to \mathbb{F}$ of <u>individual</u> degree at most δ_{lf}. The* **value** *of $\mathcal{F}$ on an input $\mathtt{x}$ is denoted $\mathcal{F}[\mathtt{x}]$ and equals $r_T[\mathtt{x}]$, which we define below.*

> *The value of a vertex v of T on an input $\mathtt{x}$ is denoted $v[\mathtt{x}]$ and is recursively defined as follows. If v is a leaf vertex, then $v[\mathtt{x}]$ equals the polynomial $\mathtt{x}_v$. If v is an internal vertex, then $v[\mathtt{x}]$ is the $\mathsf{arity}(v)$-variate polynomial over $\mathbb{F}$ defined by the following expression:*
>
> $$\sum_{\vec{\beta} \in H^{m_v}} C_v\big(\vec{X}, \vec{\beta}, u_1[\mathtt{x}](\vec{X}|_{\rho_{e_1}}, \vec{\beta}|_{\sigma_{e_1}}), \ldots, u_t[\mathtt{x}](\vec{X}|_{\rho_{e_t}}, \vec{\beta}|_{\sigma_{e_t}})\big) \ , \tag{5}$$
>
> *where $t := \mathsf{out}_T(v)$, and $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ are the outgoing edges of v. In particular, $\mathcal{F}[\mathtt{x}] = r_T[\mathtt{x}]$ is a constant in $\mathbb{F}$.*

Given a sum-product formula we can ask two types of computational problems: (*evaluation*) does a given input lead to a given output? (*satisfaction*) does there exist an input that leads to a given output? We now define each of these.

Definition 8.3 (SPFE problem)**.** *The* **sum-product formula evaluation problem** *is the following: given a sum-product formula $\mathcal{F}$, value $\mathtt{y}$, and input $\mathtt{x}$ (given as a mapping from each leaf vertex v of $\mathcal{F}$'s ari-tree T to an arithmetic circuit computing the polynomial $\mathtt{x}_v$), determine if $\mathcal{F}[\mathtt{x}] = \mathtt{y}$. This problem induces the language*

$$\mathscr{L}_{\mathrm{SPFE}} := \big\{(\mathcal{F}, \mathtt{y}, \mathtt{x}) \text{ s.t. } \mathcal{F}[\mathtt{x}] = \mathtt{y}\big\} \ .$$

(When $\mathtt{x}$ is given as above, $\mathsf{space}(\mathtt{x})$ denotes the maximum space required to evaluate any circuit in $\mathtt{x}$.)

Definition 8.4 (SPFS problem)**.** *The* **sum-product formula satisfaction problem** *is the following: given a sum-product formula $\mathcal{F}$, partial mapping of leaf vertices to arithmetic circuits $\mathtt{x}$, and value $\mathtt{y}$, determine if there exists a mapping $\mathtt{z}$ from the leaf vertices not in the domain of $\mathtt{x}$ to polynomials s.t. $\mathcal{F}[\mathtt{x}, \mathtt{z}] = \mathtt{y}$. This problem induces the relation*

$$\mathscr{R}_{\mathrm{SPFS}} := \big\{\big((\mathcal{F}, \mathtt{y}, \mathtt{x}), \mathtt{z}\big) \text{ s.t. } \mathcal{F}[\mathtt{x}, \mathtt{z}] = \mathtt{y}\big\} \ .$$

(We refer to $\mathtt{x}$ as the explicit input and $\mathtt{z}$ as the auxiliary input.)

8.2.2 Delegating sum-product formula evaluation problems

We give an Interactive Proof to delegate sum-product formula *evaluation* problems.

Theorem 8.5 (IP for SPFE)**.** *There exists a public-coin Interactive Proof for the language $\mathscr{L}_{\mathrm{SPFE}}$. In more detail:*

$$\mathscr{L}_{\mathrm{SPFE}} \in \mathbf{AM}\left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(T) \cdot |V(T)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(T) \cdot \mathsf{arity}(T)) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{F}|, |\mathtt{x}|, |H|^{\mathsf{arity}(T)}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{F}|) + O(|\mathtt{x}|) \\ \textit{verifier space:} & O(\mathsf{arity}(T) \cdot \mathsf{width}(T) \cdot \log|\mathbb{F}| + \log|\mathcal{F}| + \mathsf{space}(\mathtt{x})) \end{array}\right] .$$

Before describing the Interactive Proof system, we define for every vertex v in the ari-tree (of a sum-product formula) a function $\hat{v}[\mathtt{x}]$ based on the function $v[\mathtt{x}]$, as follows. If v is a leaf vertex, then $\hat{v}[\mathtt{x}]$ equals the polynomial $\mathtt{x}_v$. If instead v is an internal vertex, then $\hat{v}[\mathtt{x}]$ is the low-degree extension of the evaluation of $v[\mathtt{x}]$ on $H^{\mathsf{arity}(v)}$:

$$\hat{v}[\mathtt{x}](\vec{X}) := \sum_{\vec{\alpha}\in H^{\mathsf{arity}(v)}} I_{H^{\mathsf{arity}(v)}}(\vec{X},\vec{\alpha}) \sum_{\vec{\beta}\in H^{m_v}} C_v\big(\vec{\alpha},\vec{\beta}, u_1[\mathtt{x}](\vec{\alpha}|_{\rho_{e_1}},\vec{\beta}|_{\sigma_{e_1}}), \ldots, u_t[\mathtt{x}](\vec{\alpha}|_{\rho_{e_t}},\vec{\beta}|_{\sigma_{e_t}})\big) \ . \tag{6}$$

Since $\hat{v}[\mathtt{x}]$ agrees with $v[\mathtt{x}]$ on $H^{\mathsf{arity}(v)}$, we can equivalently define $\hat{v}[\mathtt{x}]$ in terms of the $\hat{u}_j[\mathtt{x}]$ rather than the $u_j[\mathtt{x}]$:

$$\begin{aligned} \hat{v}[\mathtt{x}](\vec{X}) &:= \sum_{\vec{\alpha}\in H^{\mathsf{arity}(v)}} I_{H^{\mathsf{arity}(v)}}(\vec{X},\vec{\alpha}) \sum_{\vec{\beta}\in H^{m_v}} C_v\big(\vec{\alpha},\vec{\beta}, \hat{u}_1[\mathtt{x}](\vec{\alpha}|_{\rho_{e_1}},\vec{\beta}|_{\sigma_{e_1}}), \ldots, \hat{u}_t[\mathtt{x}](\vec{\alpha}|_{\rho_{e_t}},\vec{\beta}|_{\sigma_{e_t}})\big) \\ &= \sum_{\vec{\alpha}\in H^{\mathsf{arity}(v)}} \sum_{\vec{\beta}\in H^{m_v}} I_{H^{\mathsf{arity}(v)}}(\vec{X},\vec{\alpha}) \cdot C_v\big(\vec{\alpha},\vec{\beta}, \hat{u}_1[\mathtt{x}](\vec{\alpha}|_{\rho_{e_1}},\vec{\beta}|_{\sigma_{e_1}}), \ldots, \hat{u}_t[\mathtt{x}](\vec{\alpha}|_{\rho_{e_t}},\vec{\beta}|_{\sigma_{e_t}})\big) \ . \end{aligned}$$

Note that the summand in the last line above is a polynomial, and its individual degree in $(\vec{\alpha},\vec{\beta})$ is at most $|H| + \delta_{\mathsf{in}} \cdot \max\{\delta_{\mathsf{lf}}, |H|\} \le 2\delta_{\mathsf{in}}\delta_{\mathsf{lf}}$. Indeed, $\hat{u}_i[\mathtt{x}](\vec{\alpha}|_{\rho_{e_i}},\vec{\beta}|_{\sigma_{e_i}})$ has individual degree at most δ_{lf} if u_i is a leaf, and individual degree at most $|H|$ otherwise (as $\hat{v}[\mathtt{x}]$ has individual degree at most $|H|$ in $\vec{X}$); C_v computes a polynomial of total degree at most δ_{in}; and $I_{H^{\mathsf{arity}(v)}}(\vec{X},\vec{\alpha})$ has individual degree at most $|H|$ in $\vec{\alpha}$. We use this degree bound below.

Proof. The prover and verifier receive a SPFE instance $(\mathcal{F},\mathtt{y},\mathtt{x})$ as input. They both associate, for each vertex v of its ari-tree T, a label $(\vec{\gamma}_v, a_v)$ with $\vec{\gamma}_v \in \mathbb{F}^{\mathsf{arity}(v)}$ and $a_v \in \mathbb{F}$; for the root, this label equals $(\perp,\mathtt{y})$, while for all other vertices this label is defined during the protocol. The prover and verifier then interact as follows.

1. For every internal vertex v of T taken in (any) topological order, letting $t := \mathsf{out}_T(v)$:
   (a) The prover and verifier invoke the sumcheck protocol [LFKN92; Sha92] on the claim "$\hat{v}[\mathtt{x}](\vec{\gamma}_v) = a_v$". By the end of this subprotocol, the verifier has chosen $\vec{c}_1 \in \mathbb{F}^{\mathsf{arity}(v)}$ and $\vec{c}_2 \in \mathbb{F}^{m_v}$ uniformly at random, and has derived from the prover's messages a value $b \in \mathbb{F}$ that allegedly satisfies the following equality:
   $$b := I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_v,\vec{c}_1) \cdot C_v\big(\vec{c}_1,\vec{c}_2, \hat{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}},\vec{c}_2|_{\sigma_{e_1}}), \ldots, \hat{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}},\vec{c}_2|_{\sigma_{e_t}})\big) \ , \tag{7}$$
   where $e_1 = (v,u_1), \ldots, e_t = (v,u_t)$ are v's outgoing edges.
   (b) The prover sends $h_1 := \hat{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}},\vec{c}_2|_{\sigma_{e_1}}), \ldots, h_t := \hat{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}},\vec{c}_2|_{\sigma_{e_t}}) \in \mathbb{F}$, and the verifier checks that
   $$b = I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_v,\vec{c}_1) \cdot C_v\big(\vec{c}_1,\vec{c}_2,h_1,\ldots,h_t\big) \ . \tag{8}$$
   (c) For $j = 1,\ldots,t$, the verifier sets $(\vec{\gamma}_{u_j}, a_{u_j}) := ((\vec{c}_1|_{\rho_{e_j}},\vec{c}_2|_{\sigma_{e_j}}), h_j)$.
2. For every leaf vertex v of T, the verifier checks that $\hat{v}[\mathtt{x}](\vec{\gamma}_v) = a_v$, i.e., that $\mathtt{x}_v(\vec{\gamma}_v) = a_v$.

While the above description considers sequential invocations of the sumcheck protocol, these can be run in parallel in $\mathsf{depth}(T)$ phases: first the root (which has depth 0), then all vertices of depth 1, then all vertices of depth 2, and so on until all vertices of depth $\mathsf{depth}(T)-1$. Each such phase requires $O(\mathsf{arity}(T))$ rounds, so that the number of rounds is now $O(\mathsf{depth}(T)\cdot\mathsf{arity}(T))$, as claimed. The claimed running times for the prover and verifier follow immediately from the above description. The claimed space bound follows from the observation that in phase i, if we also check the

leaf vertices at depth i during this phase, then the verifier may discard $(\vec{\gamma}_v, a_v)$ for all vertices v with $\mathsf{depth}_T(v) < i-1$; and that the value of $I_{H^{\mathsf{arity}(v)}}$ can be computed in space $O(\log(\mathsf{arity}(v)) + \log|\mathbb{F}|)$. We are left to argue the claimed soundness error.

If for some internal vertex v it holds that $v[\mathtt{x}](\gamma_v) \neq a_v$, then the soundness property of the sumcheck protocol implies that either the verifier rejects or Equation 7 holds with probability at most $2\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(v)/|\mathbb{F}|$. In this latter case, either Equation 7 fails to hold and the verifier rejects, or there exists $j \in \{1, \ldots, \mathsf{out}_T(v)\}$ such that $e_j \neq \hat{u}_j[\mathtt{x}](\vec{c}_1, \vec{c}_2)$, which means that there exists a vertex u in the next layer (in fact, $u = u_j$ suffices) for which $\hat{u}[\mathtt{x}](\vec{\gamma}_u) \neq a_u$. If u is a leaf vertex then the verifier will reject when considering u; otherwise we repeat the above argument. Taking a union bound over the internal vertices of T yields the claimed soundness error. □

8.2.3 Delegating sum-product formula satisfaction problems

We give an Interactive PCP to delegate sum-product formula *satisfaction* problems, via a simple extension of the Interactive Proof for *evaluation* problems in the previous section. Similarly to [Sha92; She92; GKR15], the verifier only needs to access the formula's input at a few locations, at the end of the protocol; thus the prover can simply send the input as a proof oracle, and the verifier can query it (via suitable low-degree testing and self-correction of polynomials).

Theorem 8.6 (IPCP for SPFS)**.** *There exists a (public-coin and non-adaptive) Interactive PCP for the relation $\mathscr{R}_{\mathrm{SPFS}}$. In more detail:*

$$\mathscr{R}_{\mathrm{SPFS}} \in \mathbf{IPCP}\left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(T) \cdot |V(T)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(T) \cdot \mathsf{arity}(T)) \\ \textit{proof length:} & O(|V(T)| \cdot |\mathbb{F}|^{\mathsf{arity}(T)}) \\ \textit{query complexity:} & |V(T)| \cdot \mathrm{poly}(\log|\mathbb{F}|, \mathsf{arity}(T), \delta_{\mathsf{lf}}) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{F}|, |\mathtt{x}|, |\mathtt{z}|, |H|^{\mathsf{arity}(T)}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{F}|, |\mathtt{x}|) \\ \textit{verifier space:} & O(\mathsf{arity}(T) \cdot \mathsf{width}(T) \cdot \log|\mathbb{F}| + \log|\mathcal{F}| + \mathsf{space}(\mathtt{x})) \end{array}\right].$$

Proof sketch. The prover and verifier receive a SPFS instance $(\mathcal{F}, \mathtt{y}, \mathtt{x})$ as input, and the prover additionally receives an auxiliary input $\mathtt{z}$ for $\mathcal{F}$ that is a valid witness for $(\mathcal{F}, \mathtt{y}, \mathtt{x})$.

- **Oracle.** The prover sends to the verifier the proof string $\pi := \mathcal{F}$, where each polynomial $\mathtt{z}_v \colon \mathbb{F}^{\mathsf{arity}(v)} \to \mathbb{F}$ is represented by its evaluation table over the whole domain.
- **Interaction.** The prover and verifier engage in an Interactive Proof for the claim "$(\mathcal{F}, \mathtt{y}, (\mathtt{x}, \mathtt{z})) \in \mathscr{L}_{\mathrm{SPFE}}$" using the protocol from the proof of Theorem 8.5 above. The verifier must access $\mathtt{z}$ only at the end of the protocol, and at few locations: for each leaf vertex v of T where v is not in the domain of $\mathtt{x}$, the verifier needs the value of $\mathtt{z}_v$ at a single location $\vec{\gamma}_v$. Thus, the verifier tests that each $\mathtt{z}_v$ is close to the evaluation of a polynomial of suitable degree [GS06; GR15], and then uses self-correction to read each $\mathtt{z}_v(\vec{\gamma}_v)$ [RS96; AS03].

Setting parameters for low-degree testing and self-correction appropriately (for the case of individual-degree multivariate polynomials) yields the parameters claimed in the theorem statement. □

8.3 Sum-product circuits

The purpose of this section is to (i) introduce *sum-product circuits*, and (ii) give proof systems for two computational problems about these, *evaluation* and *satisfaction*.

8.3.1 Formal definition

As with a boolean circuit, the 'topology' of a a sum-product circuit is a *directed acyclic multi-graph*: a tuple $G = (V, E)$ where E is a multi-set of directed edges in $V \times V$ with no directed cycles. We assume that there is a *single* vertex $r_G \in V$ with in-degree zero, known as the root. The vertices with out-degree zero are known as the leaves, while

all other vertices are known as internal vertices. We also assume that, for every vertex v, all directed paths from the root r_G to v have the same length, which we denote $\mathsf{depth}_G(v)$. The depth of G, denoted $\mathsf{depth}(G)$, is the maximum depth of any vertex v in V. The width of G, denoted $\mathsf{width}(G)$, is the maximum number of vertices at any depth: $\max_{i=1}^{\mathsf{depth}(G)} |\{v \in V : \mathsf{depth}_G(v) = i\}|$. The in-degree and out-degree of a vertex v are denoted by $\mathsf{in}_G(v)$ and $\mathsf{out}_G(v)$; we also define $\mathsf{in}(G) := \max_{v \in V} \mathsf{in}_G(v)$.

Definition 8.7. *A tuple $G = (V, E, \rho, \sigma)$ is an* **ari-graph** *if (V, E) is a directed acyclic multi-graph and both ρ and σ label every edge e in E with finite sets of positive integers ρ_e and σ_e that satisfy the following property. For every vertex v in V, there exists a (unique) non-negative integer $\mathsf{arity}(v)$ such that: (1) if v is the root then $\mathsf{arity}(v) = 0$, otherwise $\mathsf{arity}(v) = |\rho_{e_1}| + |\sigma_{e_1}| = \cdots = |\rho_{e_{\mathsf{in}_G(v)}}| + |\sigma_{e_{\mathsf{in}_G(v)}}|$ where $e_1, \ldots, e_{\mathsf{in}_G(v)}$ are v's incoming edges; (2) $\rho_{e_1}, \ldots, \rho_{e_{\mathsf{out}_G(v)}} \subseteq \{1, \ldots, \mathsf{arity}(v)\}$, where $e_1, \ldots, e_{\mathsf{out}_G(v)}$ are v's outgoing edges.*

For convenience, we denote by $\mathsf{arity}(G)$ the maximum of $\mathsf{arity}(v)$ across all vertices v in the vertex set V of G. Moreover, for every vertex v, we define $m_v := \max(\sigma_{e_1} \cup \cdots \cup \sigma_{e_{\mathsf{out}_G(v)}})$ so that $\sigma_{e_1}, \ldots, \sigma_{e_{\mathsf{out}_G(v)}} \subseteq \{1, \ldots, m_v\}$.

We are now ready to define a sum-product circuit $\mathcal{C}$, an input $\mathtt{x}$ for $\mathcal{C}$, and how to evaluate $\mathcal{C}$ on $\mathtt{x}$.

Definition 8.8. *A* **sum-product circuit** *is a tuple $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ where: $\mathbb{F}$ is a finite field, H is a subset of $\mathbb{F}$ (represented as a list of field elements), $\delta_{\mathsf{in}}, \delta_{\mathsf{lf}}$ are positive integers (represented in unary) with $\delta_{\mathsf{lf}} \geq |H|$, $G = (V, E, \rho, \sigma)$ is an ari-graph, and C labels each internal vertex v of G with an arithmetic circuit $C_v(\vec{X}, \vec{Y}, \vec{Z}) \colon \mathbb{F}^{\mathsf{arity}(v)} \times \mathbb{F}^{m_v} \times \mathbb{F}^{\mathsf{out}_T(v)} \to \mathbb{F}$ of <u>total</u> degree at most δ_{in}. An* **input** *$\mathtt{x}$ for $\mathcal{C}$ labels each leaf vertex v of G with a polynomial $\mathtt{x}_v \colon \mathbb{F}^{\mathsf{depth}_G(v) \cdot m} \to \mathbb{F}$ of <u>individual</u> degree at most δ_{lf}. The* **value** *of $\mathcal{C}$ on an input $\mathtt{x}$ is denoted $\mathcal{C}[\mathtt{x}]$ and equals $r_G[\mathtt{x}]$, which we define below.*

> *The value of a vertex v of G on an input $\mathtt{x}$ is denoted $v[\mathtt{x}]$ and is recursively defined as follows. If v is a leaf vertex, then $v[\mathtt{x}]$ equals the polynomial $\mathtt{x}_v$. If v is an internal vertex, then $v[\mathtt{x}]$ is the $\mathsf{arity}(v)$-variate polynomial over $\mathbb{F}$ defined by the following expression:*
>
> $$\sum_{\vec{\beta} \in H^{m_v}} C_v\big(\vec{X}, \vec{\beta}, u_1[\mathtt{x}](\vec{X}|_{\rho_{e_1}}, \vec{\beta}|_{\sigma_{e_1}}), \ldots, u_t[\mathtt{x}](\vec{X}|_{\rho_{e_t}}, \vec{\beta}|_{\sigma_{e_t}})\big) \ ,$$
>
> *where $t := \mathsf{out}_G(v)$, and $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ are the outgoing edges of v (with multiplicity). In particular, $\mathcal{C}[\mathtt{x}] = r_G[\mathtt{x}]$ is a constant in $\mathbb{F}$.*

Given a sum-product circuit we can ask two types of computational problems: (*evaluation*) does a given input lead to a given output? (*satisfaction*) does there exist an input that leads to a given output? We now define each of these.

Definition 8.9 (SPCE problem)**.** *The* **sum-product circuit evaluation problem** *is the following: given a sum-product circuit $\mathcal{C}$, value $\mathtt{y}$, and input $\mathtt{x}$ (given as a mapping from each leaf vertex v of $\mathcal{C}$'s ari-graph G to an arithmetic circuit computing the polynomial $\mathtt{x}_v$), determine if $\mathcal{C}[\mathtt{x}] = \mathtt{y}$. This problem induces the language*

$$\mathscr{L}_{\mathrm{SPCE}} := \big\{(\mathcal{C}, \mathtt{y}, \mathtt{x}) \text{ s.t. } \mathcal{C}[\mathtt{x}] = \mathtt{y}\big\} \ .$$

(When $\mathtt{x}$ is given as above, $\mathsf{space}(\mathtt{x})$ denotes the maximum space required to evaluate any circuit in $\mathtt{x}$.)

Definition 8.10 (SPCS problem)**.** *The* **sum-product circuit satisfaction problem** *is the following: given a sum-product circuit $\mathcal{C}$, partial mapping of leaf vertices to arithmetic circuits $\mathtt{x}$, and value $\mathtt{y}$, determine if there exists a mapping $\mathtt{z}$ from the leaf vertices not in the domain of $\mathtt{x}$ to polynomials s.t. $\mathcal{C}[\mathtt{x}, \mathtt{z}] = \mathtt{y}$. This problem induces the relation*

$$\mathscr{R}_{\mathrm{SPCS}} := \big\{\big((\mathcal{C}, \mathtt{y}, \mathtt{x}), \mathtt{z}\big) \text{ s.t. } \mathcal{C}[\mathtt{x}, \mathtt{z}] = \mathtt{y}\big\} \ .$$

(We refer to $\mathtt{x}$ as the explicit input and $\mathtt{z}$ as the auxiliary input.)

8.3.2 Delegating sum-product circuit evaluation problems

We give an Interactive Proof to delegate sum-product formula *evaluation* problems.

Theorem 8.11 (IP for SPCE)**.** *There exists a public-coin Interactive Proof for the language $\mathscr{L}_{\mathrm{SPCE}}$. In more detail:*

$$\mathscr{L}_{\mathrm{SPCE}} \in \mathbf{AM}\left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(G) \cdot |V(G)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(G) \cdot \mathsf{arity}(G)) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{C}|, |\mathtt{x}|, |H|^{\mathsf{arity}(G)}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{C}|) + O(\mathsf{in}(G) \cdot |\mathtt{x}|) \\ \textit{verifier space:} & O(\mathsf{arity}(G) \cdot \mathsf{width}(G) \cdot \mathsf{in}(G) \cdot \log|\mathbb{F}| + \log|\mathcal{C}| + \mathsf{space}(\mathtt{x})) \end{array}\right].$$

Proof sketch. The prover and verifier receive as input a SPCE instance $(\mathcal{C}, \mathtt{y}, \mathtt{x})$. They both associate with each vertex v of its ari-graph G a *set* of labels L_v; for the root, this set contains only the pair $(\perp, \mathtt{y})$, while for all other vertices this set is initially empty and will be populated with at most $\mathsf{in}_G(v)$ pairs during the protocol. The prover and verifier then interact as follows.

1. For every internal vertex v of G taken in (any) topological order, letting $t := \mathsf{out}_G(v)$:
   (a) For every $(\vec{\gamma}_j, a_j)$ in L_v, the verifier samples a random $\alpha_j \in \mathbb{F}$ and sends it to the prover.
   (b) The prover and verifier invoke the sumcheck protocol on the following claim:

$$\text{“} \quad \sum_{j=1}^{|L_v|} \alpha_j \hat{v}[\mathtt{x}](\vec{\gamma}_j) = \sum_{j=1}^{|L_v|} \alpha_j a_j \quad \text{”}.$$

   By the end of this subprotocol, the verifier has chosen $\vec{c}_1 \in \mathbb{F}^{\mathsf{arity}(v)}$ and $\vec{c}_2 \in \mathbb{F}^{m_v}$ uniformly at random, and has derived from the prover's messages a value $b \in \mathbb{F}$ that allegedly satisfies the following equality:

$$b = \sum_{j=1}^{|L_v|} \alpha_j \Big(I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v\big(\vec{c}_1, \vec{c}_2, \hat{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}}, \vec{c}_2|_{\sigma_{e_1}}), \ldots, \hat{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}}, \vec{c}_2|_{\sigma_{e_t}})\big)\Big) \ , \tag{9}$$

   where $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ are the outgoing edges of v (with multiplicity).
   (c) The prover sends $h_1 := \hat{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}}, \vec{c}_2|_{\sigma_{e_1}}), \ldots, h_t := \hat{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}}, \vec{c}_2|_{\sigma_{e_t}}) \in \mathbb{F}$, and the verifier checks that

$$b = \sum_{j=1}^{|L_v|} \alpha_j \Big(I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v\big(\vec{c}_1, \vec{c}_2, h_1, \ldots, h_t\big)\Big) \ . \tag{10}$$

   (d) For every $j = 1, \ldots, t$, the verifier adds the label $((\vec{c}_1|_{\rho_{e_j}}, \vec{c}_2|_{\sigma_{e_j}}), h_j)$ to L_{u_j}.

2. For every leaf vertex v of G, and for every $(\vec{\gamma}, a) \in L_v$, the verifier checks that $\hat{v}[\mathtt{x}](\vec{\gamma}) = a$, i.e., that $\mathtt{x}_v(\vec{\gamma}) = a$. (Note that every set L_v has had $\mathsf{in}_G(v)$ labels added to it. The size of L_v is then at most $\mathsf{in}_G(v)$, with the 'strictly less' case occurring if the verifier happens to have added the same label twice.)

While the above description considers sequential invocations of the sumcheck protocol, these can be run in parallel in $\mathsf{depth}(G)$ phases: first the root (which has depth 0), then all vertices of depth 1, then all vertices of depth 2, and so on until all vertices of depth $\mathsf{depth}(G) - 1$. Each such phase requires $O(\mathsf{arity}(G))$ rounds, so that the number of rounds is now $O(\mathsf{depth}(G) \cdot \mathsf{arity}(G))$, as claimed. The claimed running times for the prover and verifier follow immediately from the above description. The claimed space bound can be attained by discarding all labels at previous levels before moving to the next level, and checking leaf vertices at the same time as the internal vertices at the same depth. We are left to argue the claimed soundness error.

Suppose that, when considering some internal vertex v of G in the protocol above, there exists $(\vec{\gamma}, a) \in L_v$ such that $\hat{v}[\mathtt{x}](\vec{\gamma}) \neq a$. Then, with probability at least $1 - 1/|\mathbb{F}|$, it holds that $\sum_{j=1}^{|L_v|} \alpha_j \hat{v}[\mathtt{x}](\vec{\gamma}_j) \neq \sum_{j=1}^{|L_v|} \alpha_j a_j$, which means that the prover and verifier invoke the sumcheck protocol on a false claim. By the soundness of the sumcheck protocol, either

the verifier rejects or Equation 9 holds with probability at most $2\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(v)/|\mathbb{F}|$. In this latter case, either Equation 10 fails to hold and the verifier rejects, or there exists $j \in \{1, \ldots, \mathsf{out}_G(v)\}$ such that $h_j \neq \hat{u}_j[\mathtt{x}](\vec{c}_1|_{\rho_{e_j}}, \vec{c}_2|_{\sigma_{e_j}})$, which means that there exists a vertex u in the next layer (in particular, $u = u_j$) for which there exists $(\vec{\gamma}', a') \in L_u$ such that $\hat{u}[\mathtt{x}](\vec{\gamma}') \neq a'$. If u is a leaf vertex then the verifier will reject when considering u; otherwise we repeat the above argument. Taking a union bound over the internal vertices of G yields the claimed soundness error. □

8.3.3 Delegating sum-product circuit satisfaction problems

We give an Interactive PCP to delegate sum-product circuit *satisfaction* problems.

Theorem 8.12 (IPCP for SPCS)**.** *There exists a (public-coin and non-adaptive) Interactive PCP for the relation $\mathscr{R}_{\mathrm{SPCS}}$. In more detail:*

$$\mathscr{R}_{\mathrm{SPCS}} \in \mathbf{IPCP}\left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(G) \cdot |E(G)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(G) \cdot \mathsf{arity}(G)) \\ \textit{proof length:} & O(|V(G)| \cdot |\mathbb{F}|^{\mathsf{arity}(G)}) \\ \textit{query complexity:} & |V(G)| \cdot \mathrm{poly}(\log|\mathbb{F}|, \mathsf{arity}(G), \delta_{\mathsf{lf}}) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{C}|, |\mathtt{x}|, |\mathtt{z}|, |H|^{\mathsf{arity}(G)}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{C}|, |\mathtt{x}|) \\ \textit{verifier space:} & O(\mathsf{arity}(G) \cdot \mathsf{width}(G) \cdot \mathsf{in}(G) \cdot \log|\mathbb{F}| + \log|\mathcal{C}| + \mathsf{space}(\mathtt{x})) \end{array}\right].$$

Proof. The protocol is analogous to that in the proof of Theorem 8.6, which considers the relation $\mathscr{R}_{\mathrm{SPFS}}$ (sum-product formula satisfaction) rather than $\mathscr{R}_{\mathrm{SPCS}}$ (sum-product circuit satisfaction). Specifically, we only need to replace the Interactive Proof for the language $\mathscr{L}_{\mathrm{SPFE}}$ (sum-product formula evaluation) with the Interactive Proof for the language $\mathscr{L}_{\mathrm{SPCE}}$ (sum-product circuit evaluation) that we gave in the proof of Theorem 8.11. We omit the details, except for one technicality that we now describe.

The strategy described in the above paragraph eventually leads the verifier to read, via self-correction, at most $\mathsf{in}_G(v)$ values of $\mathtt{z}_v$ for every leaf vertex v of G not in the domain of $\mathtt{x}$. Overall, the verifier reads at most $\sum_v \mathsf{in}_G(v) \leq |E(G)|$ values via self-correction, which corresponds to $|E(G)| \cdot \mathrm{poly}(\log|\mathbb{F}| + \mathsf{arity}(G) + \delta_{\mathsf{lf}})$ actual queries, and a soundness error of $O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(G) \cdot |V(G)|/|\mathbb{F}|)$. To obtain the stated query complexity, we reduce the number of values read via self-correction to a single value per leaf vertex, via the following standard trick.

Let $t_1, \ldots, t_{\mathsf{in}_G(v)} \in \mathbb{F}$ be arbitrary distinct values known to both the prover and verifier, and let $A_v \colon \mathbb{F} \to \mathbb{F}^{\mathsf{arity}(v)}$ be the unique polynomial of degree less than $\mathsf{in}_G(v)$ such that $A_v(t_i) = \vec{\gamma}_i$ for every $(\vec{\gamma}_i, a_i) \in L_v$. The prover sends $B_v := (\mathtt{z}_v \circ A_v) \colon \mathbb{F} \to \mathbb{F}$ to the verifier (as a list of at most $\mathsf{arity}(v) \cdot \mathsf{in}_G(v) \cdot \delta_{\mathsf{lf}}$ coefficients), who checks that $B_v(t_i) = a_i$ for every $i \in \{1, \ldots, \mathsf{in}_G(v)\}$. The verifier then picks $t \in \mathbb{F}$ uniformly at random and checks that $\mathtt{z}_v(A_v(t)) = B_v(t)$; this involves obtaining, via self-correction, the value of $\mathtt{z}_v$ at $A_v(t)$. Soundness is maintained because if the prover sends $B'_v \neq B_v$ then the probability that $\mathtt{z}_v(A_v(t)) = B'_v(t)$ for uniformly random $t \in \mathbb{F}$ is at most $\mathsf{arity}(v) \cdot \mathsf{in}_G(v) \cdot \delta_{\mathsf{lf}}/|\mathbb{F}|$. The overall soundness error is then, by a union bound, at most $O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(G) \cdot |V(G)|/|\mathbb{F}|) + \sum_v \mathsf{arity}(v) \cdot \mathsf{in}_G(v) \cdot \delta_{\mathsf{lf}}/|\mathbb{F}| = O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{arity}(G) \cdot |E(G)|/|\mathbb{F}|)$. The additional space required for this test is $O(\mathsf{arity}(G) \cdot \mathsf{in}(G) \cdot \log|\mathbb{F}| + \log|\mathcal{C}|)$, so the space bound is unaffected. □

9 Zero knowledge sum-product protocols

The protocols for delegating sum-product computations described in Section 8 are not zero knowledge. We show how to delegate, in the Interactive PCP model, sum-product circuit evaluation problems (Section 9.1) and satisfaction problems (Section 9.2). As a special case, we also obtain the same for sum-product formulas.

9.1 The case of sum-product evaluation

The purpose of this section is to show that the language $\mathscr{L}_{\mathrm{SPCE}}$ (consisting of sum-product circuit evaluation problems, see Definition 8.9) has perfect zero knowledge Interactive PCPs:

Theorem 9.1 (PZK IPCP for $\mathscr{L}_{\mathrm{SPCE}}$)**.** *For every query bound function $b(n)$, the language $\mathscr{L}_{\mathrm{SPCE}}$ has a (public-coin and non-adaptive) Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail, letting $\alpha := \log b / \log |H|$:*

$$\mathscr{L}_{\mathrm{SPCE}} \in \mathbf{PZK\text{-}IPCP} \left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{in}(G) \cdot (\mathsf{arity}(G)+\alpha) \cdot |V(G)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(G) \cdot (\mathsf{arity}(G)+\alpha)) \\ \textit{proof length:} & O(|V(G)| \cdot |\mathbb{F}|^{\mathsf{arity}(G)+\alpha}) \\ \textit{query complexity:} & |V(G)| \cdot \mathrm{poly}(\log|\mathbb{F}|, \mathsf{arity}(G), \alpha, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, \mathsf{in}(G)) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{C}|, |\mathbb{x}|, |\mathbb{F}|^{\mathsf{arity}(G)+\alpha}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{C}|, \alpha) + O(\mathsf{in}(G) \cdot |\mathbb{x}|) \\ \textit{verifier space:} & O((\mathsf{arity}(G)+\alpha) \cdot \mathsf{width}(G) \cdot \mathsf{in}(G) \cdot \log|\mathbb{F}| + \log|\mathcal{C}| + \mathsf{space}(\mathbb{x})) \\ \textit{simulator overhead:} & \mathrm{poly}(|\mathcal{C}|, \alpha) \cdot (|\mathbb{x}| + \mathsf{q}_V^3). \end{array} \right] .$$

We introduce some notation before the proof. Given a subset H of $\mathbb{F}$ and a positive integer m, we denote by $\mathbb{Z}_{H^m}$ the m-variate polynomial $\prod_{i=1}^{m}\prod_{\alpha\in H}(X_i - \alpha)$. Note that $\mathbb{Z}_{H^m}$ is zero on H^m and nonzero on $(\mathbb{F}-H)^m$, and can be evaluated in $\mathrm{poly}(|H|+m)$ field operations and space $O(\log m + \log|\mathbb{F}|)$.

To guide us to the proof of the above theorem, it is instructive to look at why the protocol of Theorem 8.11 may not be zero knowledge. We identify two potential sources of leakage: the first is the values $\hat{v}[\mathbb{x}](\vec{c})$ for internal vertices v which the prover sends in Step 1c of the protocol; the second is the partial sums which are leaked by the sumcheck subprotocol itself.

We resolve the first issue by replacing, for each internal vertex v, the low-degree extension $\hat{v}[\mathbb{x}]$ in the SPCE protocol with a *randomized* low-degree extension $\dot{v}[\mathbb{x}]$, which is $\mathsf{in}_G(v)$-wise independent outside of $H^{\mathsf{arity}(v)}$. More precisely, for any list of distinct query points $\vec{\gamma}_1, \ldots, \vec{\gamma}_{\mathsf{in}_G(v)} \in (\mathbb{F}-H)^{\mathsf{arity}(v)}$, $(\dot{v}[\mathbb{x}](\vec{\gamma}_i))_{i=1}^{\mathsf{in}_G(v)}$ is uniformly random in $\mathbb{F}^\ell$. Given such a low-degree extension, it suffices to ensure that the verifier may only learn its evaluations inside $(\mathbb{F}-H)^m$, and then only on at most $\mathsf{in}_G(v)$ distinct points; then we can simulate all of these queries with uniformly random field elements.

Given a sum-product circuit $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$, we define for every vertex v in its ari-graph $G = (V, E, \rho, \sigma)$ a *random variable* $\dot{v}[\mathbb{x}]$ based on $v[\mathbb{x}]$, as follows. Let $\lambda := 2\delta_{\mathsf{in}} \cdot (\delta_{\mathsf{lf}} + \mathsf{in}(G)) = \Omega(|H|)$. If v is a leaf vertex, then $\dot{v}[\mathbb{x}]$ simply equals $\mathbb{x}_v$ with probability 1. If instead v is an internal vertex, then $\dot{v}[\mathbb{x}]$ is a "randomized" low-degree extension of $v[\mathbb{x}]$:

$$\dot{v}[\mathbb{x}](\vec{X}) := \hat{v}[\mathbb{x}](\vec{X}) + \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{X}) \sum_{\vec{\gamma}\in G^k} R_v(\vec{X}, \vec{\gamma}) \ ,$$

where $\hat{v}[\mathbb{x}]$ is the *fixed* low-degree extension of $v[\mathbb{x}]$ that we used in Section 8 (see Equation 6), k is a security parameter, G is an arbitrary subset of $\mathbb{F}$ of size λ with $0 \in G$, R_v is uniformly random in $\mathbb{F}[X_{1,\ldots,\mathsf{arity}(v)}^{\leq \mathsf{in}_G(v)}, Y_{1,\ldots,k}^{\leq 2\lambda}]$, and $\mathsf{in}_G(v)$ is the in-degree of v in G. (In the protocol, the verifier receives in the oracle message the evaluation table of independently drawn R_v, for every internal vertex v of G.)

Since $\dot{v}[\mathtt{x}]$ agrees with $v[\mathtt{x}]$ on $H^{\mathsf{arity}(v)}$, we can equivalently write (using the definition of $\hat{v}[\mathtt{x}]$):

$$\dot{v}[\mathtt{x}](\vec{X}) := \left(\sum_{\vec{\alpha} \in H^{\mathsf{arity}(v)}} \sum_{\vec{\beta} \in H^{m_v}} I_{H^{\mathsf{arity}(v)}}(\vec{X}, \vec{\alpha}) \cdot C_v\big(\vec{\alpha}, \vec{\beta}, \dot{u}_1[\mathtt{x}](\vec{\alpha}|_{\rho_{e_1}}, \vec{\beta}|_{\sigma_{e_1}}), \ldots, \dot{u}_t[\mathtt{x}](\vec{\alpha}|_{\rho_{e_t}}, \vec{\beta}|_{\sigma_{e_t}})\big) \right)$$
$$+ \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{X}) \sum_{\vec{\gamma} \in G^k} R_v(\vec{X}, \vec{\gamma})$$
$$= \sum_{\vec{\alpha} \in H^{\mathsf{arity}(v)}} \sum_{\vec{\beta} \in H^{m_v}} \sum_{\vec{\gamma} \in G^k} \Big(I_{G^k}(\vec{0}, \vec{\gamma}) \cdot I_{H^{\mathsf{arity}(v)}}(\vec{X}, \vec{\alpha}) \cdot C_v\big(\vec{\alpha}, \vec{\beta}, \dot{u}_1[\mathtt{x}](\vec{\alpha}|_{\rho_{e_1}}, \vec{\beta}|_{\sigma_{e_1}}), \ldots, \dot{u}_t[\mathtt{x}](\vec{\alpha}|_{\rho_{e_t}}, \vec{\beta}|_{\sigma_{e_t}})\big)$$
$$+ I_{H^{\mathsf{arity}(v)+m_v}}((\vec{\alpha}, \vec{\beta}), \vec{0}) \cdot \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{X}) \cdot R_v(\vec{X}, \vec{\gamma}) \Big) \ .$$

Note that the individual degree of $\dot{v}[\mathtt{x}](\vec{X})$ is exactly $|H| + \mathsf{in}_G(v)$. The individual degree of the summand in the last line (in $\vec{\alpha}, \vec{\beta}, \vec{\gamma}$) is at most $\max\{2\lambda, |H| + \mathsf{in}_G(v) + \delta_{\mathsf{in}} \cdot \max\{\delta_{\mathsf{lf}}, |H| + \max_{1 \le i \le \mathsf{out}_G(v)} \mathsf{in}_G(u_i)\}\} \le 2\lambda$.

Observe first that R_v is a perfectly-hiding commitment to the random polynomial $S_v(\vec{X}) := \sum_{\vec{\gamma} \in G^k} R_v(\vec{X}, \vec{\gamma})$, and so $S_v(\vec{x})$ itself is uniformly random even conditioned on strictly fewer than λ^k queries to R_v. Then since $\mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{X})$ is non-zero in $(\mathbb{F} - H)^{\mathsf{arity}(v)}$ and the individual degree of S_v is $\mathsf{in}_G(v)$, the required independence property holds. If we ensure that the verifier chooses its challenges in the sumcheck protocol from $\mathbb{F} - H$ rather than all of $\mathbb{F}$ (i.e. the prover aborts otherwise), then the values sent to the verifier in the SPCE protocol will indeed be uniformly random.

We resolve the second issue by replacing the sumcheck subprotocol with the zero knowledge IPCP for sumcheck defined in Section 6. This requires sending $O(|V(G)|)$ proofs, which we can concatenate together with the R_v into a single oracle. Note that here we will require the full strength of the zero knowledge guarantee which we obtain in Theorem 6.4 (as opposed to the weaker guarantee of [BCFGRS16]), because the simulator is not able to make an arbitrary polynomial number of queries to $\dot{v}[\mathtt{x}]$ for any internal vertex v. Instead the number of queries must be bounded by $\mathsf{in}_G(v)$ for each v, so that these queries can be simulated by choosing uniformly random field elements; the zero knowledge guarantee of Theorem 6.4 allows us to do exactly that.

We are now ready to put everything together.

Proof. Fix $k := \lceil \log b / \log \lambda \rceil$. The prover and verifier receive as input a SPCE instance $(\mathcal{C}, \mathtt{y}, \mathtt{x})$. We first describe the oracle message that is first sent to the verifier, and then describe the subsequent interaction between the prover and verifier.

- **Oracle.** The prover sends to the verifier a proof string π that contains, for each internal vertex v of G:
  - the evaluation table of the polynomial $R_v \in \mathbb{F}[X_{1,\ldots,\mathsf{arity}(v)}^{\le \mathsf{in}_G(v)}, Y_{1,\ldots,k}^{\le 2\lambda}]$ drawn independently and uniformly at random;
  - a proof string π_v which is the oracle sent in a λ^k-strong zero knowledge sumcheck protocol on input $(\mathbb{F}, \mathsf{arity}(v) + m_v + k, \lambda, H, \cdot)$. [3]

- **Interaction.** The prover and verifier associate, for each vertex v of G, a *set* of labels L_v; for the root, this set contains only the pair $(\perp, \mathtt{y})$, while for all other vertices this set is initially empty and will be populated during the protocol. The prover and verifier then interact as follows.

  1. For every internal vertex v of G taken in (any) topological order, letting $t := \mathsf{out}_G(v)$:
     (a) For every $(\vec{\gamma}_j, a_j)$ in L_v, the verifier samples a random $\alpha_j \in \mathbb{F}$ and sends it to the prover.
     (b) The prover and verifier invoke a λ^k-strong perfect zero knowledge Interactive Probabilistically Checkable Proof system for sumcheck (see Section 6) on the claim

$$\text{“} \quad \sum_{j=1}^{|L_v|} \alpha_j \dot{v}[\mathtt{x}](\vec{\gamma}_j) = \sum_{j=1}^{|L_v|} \alpha_j a_j \quad \text{”}$$

[3] Recall that we do not need to specify a until later on in the protocol, and it will depend on the verifier's random choices.

using π_v as the oracle, and with $I := \mathbb{F} \setminus H$. By the end of this subprotocol, the verifier has chosen $\vec{c}_1 \in \mathbb{F}^{\mathsf{arity}(v)}$, $\vec{c}_2 \in \mathbb{F}^m$, and $\vec{c}_3 \in \mathbb{F}^k$ uniformly at random, and has derived from the prover's messages a value $b \in \mathbb{F}$ that allegedly satisfies the following equality:

$$b = \sum_{j=1}^{|L_v|} \alpha_j \Big(I_{G^k}(\vec{0}, \vec{c}_3) \cdot I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v\big(\vec{c}_1, \vec{c}_2, \dot{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}}, \vec{c}_2|_{\sigma_{e_1}}), \ldots, \dot{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}}, \vec{c}_2|_{\sigma_{e_t}})\big) \\ + I_{H^{\mathsf{arity}(v)+m_v}}((\vec{c}_1, \vec{c}_2), \vec{0}) \cdot \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j) \cdot R_v(\vec{\gamma}_j, \vec{c}_3) \Big) \ , \tag{11}$$

where $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ are the outgoing edges of v (with multiplicity).

(c) The prover sends $\vec{h}_v := (h_1, \ldots, h_t)$ where $h_1 := \dot{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}}, \vec{c}_2|_{\sigma_{e_1}}), \ldots, h_t := \dot{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}}, \vec{c}_2|_{\sigma_{e_t}})$. For every $j = 1, \ldots, t$, the verifier adds the label $((\vec{c}_1|_{\rho_{e_j}}, \vec{c}_2|_{\sigma_{e_j}}), h_j)$ to L_{u_j}.

2. For every internal vertex v of G, the verifier checks that

$$b = \sum_{j=1}^{|L_v|} \alpha_j \Big(I_{G^k}(\vec{0}, \vec{c}_3) \cdot I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v\big(\vec{c}_1, \vec{c}_2, \vec{h}_v\big) + I_{H^{\mathsf{arity}(v)+m_v}}((\vec{c}_1, \vec{c}_2), \vec{0}) \cdot \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j) \cdot R_v(\vec{\gamma}_j, \vec{c}_3) \Big) \ . \tag{12}$$

For this, the prover sends $\big(R_v(\vec{\gamma}_j, \vec{c}_3)\big)_{j=1}^{|L_v|}$, which the verifier uses to compute the above expression. The verifier checks that these values are correct using a standard interpolation trick and a single query to R_v. To make this query, the verifier (i) tests that R_v is close to (the evaluation of) a polynomial in $\mathbb{F}[X_{1,\ldots,\mathsf{arity}(v)}^{\leq \mathsf{in}_G(v)}, Y_{1,\ldots,k}^{\leq 2\lambda}]$; (ii) uses self-correction to make the required query.

3. For every leaf vertex v of G, and for every $(\vec{\gamma}, a) \in L_v$, the verifier checks that $\dot{v}[\mathtt{x}](\vec{\gamma}) = a$, i.e., that $\mathtt{x}_v(\vec{\gamma}) = a$.

Efficiency. The protocol runs a λ^k-strong zero knowledge sumcheck protocol at most $|V|$ times on polynomials of at most $\mathsf{arity}(G) + k$ variables. Inspection of the protocol shows that we can execute Step 1 in parallel for all vertices at the same depth, so the number of rounds is at most $\mathsf{depth}(G)(\mathsf{arity}(G) + 2k + 2)$. The running time of the verifier is clearly polynomial in $|\mathcal{C}|$. The number of queries to the R_v is at most $|V(G)| \cdot \mathrm{poly}(\log |\mathbb{F}| + k + \lambda)$, and the number of queries to the π_v is at most $|V(G)| \cdot \mathrm{poly}(\log |\mathbb{F}| + \mathsf{arity}(G) + k + \lambda)$.

Completeness. Perfect completeness is clear from the protocol description and the perfect completeness of the zero-knowledge sumcheck protocol.

Soundness. Let $v_1, v_2, \ldots$ be any topological order of the internal vertices of G, and define let $V_{\geq i}$ be the union of $\{v_j : j \geq i\}$ and the leaf vertices of G. First we argue that if before iteration i of Step 1 there exists $v \in V_{\geq i}$ such that L_v contains $(\vec{\gamma}, a)$ with $\dot{v}[\mathtt{x}](\vec{\gamma}) \neq a$, then after this iteration with high probability either (a) the verifier rejects, or (b) there exists $w \in V_{\geq i+1}$, such that there is $(\vec{\gamma}', b) \in L_w$ with $\dot{w}[\mathtt{x}](\vec{\gamma}') \neq b$. If $v \in V_{\geq i+1}$ then there is nothing to do, so we may assume that $v = v_i$. The probability that $\sum_{j=1}^{|L_{v_i}|} \alpha_j \dot{v}_i[\mathtt{x}](\vec{\gamma}_j) = \sum_{j=1}^{|L_{v_i}|} \alpha_j a_j$ is $1/|\mathbb{F}|$. So with probability $1 - 1/|\mathbb{F}|$ we run the zero knowledge sumcheck protocol on a false claim about a polynomial with $(\mathsf{arity}(v) + m_v + k)$ variables and individual degree at most 2λ, which means that with probability at least $1 - O((\mathsf{arity}(v) + m_v + k) \cdot \lambda/(|\mathbb{F}| - |H|))$ either the verifier rejects or outputs the *false* claim

$$b = \sum_{j=1}^{|L_v|} \big(\alpha_j I_{G^k}(\vec{0}, \vec{c}_3) \cdot I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v\big(\vec{c}_1, \vec{c}_2, \dot{u}_1[\mathtt{x}](\vec{c}_1|_{\rho_{e_1}}, \vec{c}_2|_{\sigma_{e_1}}), \ldots, \dot{u}_t[\mathtt{x}](\vec{c}_1|_{\rho_{e_t}}, \vec{c}_2|_{\sigma_{e_t}})\big) \\ + I_{H^{\mathsf{arity}(v)+m_v}}((\vec{c}_1, \vec{c}_2), \vec{0}) \cdot \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j) \cdot R_v(\vec{\gamma}_j, \vec{c}_3)\big) \ .$$

The verifier receives values $(r_j)_{j=1}^{|L_v|}$, which it substitutes for $R_v(\vec{\gamma}_j, \vec{c}_3)$ in Equation 12. If this expression does not evaluate to b, then the verifier rejects. If R_v is far from any polynomial in $\mathbb{F}[X_{1,\ldots,\mathsf{arity}(v)}^{\leq \mathsf{in}_G(v)}, Y_{1,\ldots,k}^{\leq 2}]$, then the verifier rejects with high probability. If there exists some j such that $r_j \neq R_v(\vec{\gamma}_j, \vec{c}_3)$, then by the soundness of the polynomial interpolation test, the verifier will reject with probability at least $1 - O((\mathsf{arity}(v) + k) \cdot \lambda/|\mathbb{F}|)$. Otherwise, it must be

the case that $\dot{u}_j[\mathtt{x}](\vec{c}) \neq h_j$ where u_j is one of the $\mathsf{out}_G(v_i)$ children of v_i. Since $u_j \in V_{\geq i+1}$, with high probability after this iteration either the verifier has already rejected or there is $w \in V_{\geq i+1}$ and $(\vec{\gamma}', a') \in L_w$ with $\dot{w}[\mathtt{x}](\vec{\gamma}') \neq a'$.

The above implies soundness in a straightforward way: if $\mathcal{C}[\mathtt{x}] \neq \mathtt{y}$ then the condition is satisfied before the first iteration, and in each iteration with high probability either the condition is maintained or the verifier rejects. Thus with high probability either the verifier rejects or the invariant holds before the last iteration; after the last iteration, the verifier will reject with high probability because only leaf vertices are left (and claims about them are checked directly). More precisely, by a union bound over all the internal vertices, and setting the parameters of the proximity test appropriately, if $\mathcal{C}[\mathtt{x}] \neq \mathtt{y}$ then the verfier accepts with probability $O(|V(G)|(\mathsf{arity}(G) + k) \cdot \lambda/(|\mathbb{F}| - |H|))$.

Zero knowledge. We prove that the protocol has perfect zero knowledge by exhibiting a polynomial-time simulator that perfectly samples the view of any malicious verifier. We will assume that the simulator maintains the label sets L_v in the same way as the honest verifier, but for clarity we will not state this in its description.

1. For every internal vertex v of G, sample $R^v_{\mathrm{sim}} \in \mathbb{F}[X^{\leq \mathsf{in}_G(v)}_{1,\ldots,\mathsf{arity}(v)}, Y^{\leq 2\lambda}_{1,\ldots,k}]$ uniformly at random. Use R^v_{sim} to answer queries to R_v.
2. For every internal vertex v of G, run the λ^k-strong ZK sumcheck simulator on input $(\mathbb{F}, \mathsf{arity}(v) + m_v + k, \lambda, H, \cdot)$, and use it to answer queries to π_v throughout. Recall that the behavior of each simulator does not depend on the claim being proven until after the first simulated message, so we can choose these later.
3. For every internal vertex v of G taken in (any) topological order, letting $t := \mathsf{out}_G(v)$:
   (a) Receive $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{|L_v|}$ from the verifier.
   (b) Using the subsimulator for v, simulate the strong ZK sumcheck protocol on the claim

$$\text{“} \quad \sum_{j=1}^{|L_v|} \tilde{\alpha}_j \dot{v}[\mathtt{x}](\vec{\gamma}_j) = \sum_{j=1}^{|L_v|} \tilde{\alpha}_j a_j \quad \text{”}.$$

   The subsimulator will query the oracle F at a single location $\vec{c} = (\vec{c}_1, \vec{c}_2, \vec{c}_3)$ with $\vec{c}_1 \in \mathbb{F}^{\mathsf{arity}(v)}$, $\vec{c}_2 \in \mathbb{F}^{m_v}$, and $\vec{c}_3 \in \mathbb{F}^k$. Reply with the value

$$\sum_{j=1}^{|L_v|} \alpha_j \Big(I_{G^k}(\vec{0}, \vec{c}_3) \cdot I_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j, \vec{c}_1) \cdot C_v(\vec{c}_1, \vec{c}_2, h^1_{\mathrm{sim}}, \ldots, h^t_{\mathrm{sim}}) + I_{H^{\mathsf{arity}(v)+m_v}}((\vec{c}_1, \vec{c}_2), \vec{0}) \cdot \mathbb{Z}_{H^{\mathsf{arity}(v)}}(\vec{\gamma}_j) \cdot R_v(\vec{\gamma}_j, \vec{c}_3) \Big) \ ,$$

   where $h^1_{\mathrm{sim}}, \ldots, h^t_{\mathrm{sim}} \in \mathbb{F}$ are chosen as follows. Let $e_1 = (v, u_1), \ldots, e_t = (v, u_t)$ be the outgoing edges of v (with multiplicity). For every $k \in \{1, \ldots, t\}$, letting $\vec{c}_k := (\vec{c}_1|_{\rho_{e_k}}, \vec{c}_2|_{\sigma_{e_k}})$:
      i. if u_k is a leaf vertex, then $h^k_{\mathrm{sim}} := \mathtt{x}_{u_k}(\vec{c}_k)$.
      ii. if u_k is an internal vertex and $(\vec{c}_k, h) \in L_{u_k}$ for some $h \in \mathbb{F}$, $h^k_{\mathrm{sim}} := h$.
      iii. if u_k is an internal vertex and $(\vec{c}_k, h) \notin L_{u_k}$ for all $h \in \mathbb{F}$, sample h^k_{sim} at random and add $(\vec{c}_k, h^k_{\mathrm{sim}})$ to L_{u_k}.
   (c) Send $h^1_{\mathrm{sim}}, \ldots, h^t_{\mathrm{sim}}$ to the verifier.

The view of the verifier in a real execution is composed of the messages from the prover during each sumcheck protocol, the values $\dot{v}[\mathtt{x}](\vec{\gamma})$ for every internal vertex v in V and $(\vec{\gamma}, a) \in L_v$, and the verifier's queries to the oracles R_v and π_v for every internal vertex v in V.

Any b-query malicious verifier $\tilde{V}$ may query any R_v at strictly fewer than b points. By Corollary 5.3, $S_v(\vec{x}) := \sum_{\vec{y} \in G^k} R_v(\vec{x}, \vec{y})$ is uniformly random in $\mathbb{F}[X^{\leq \mathsf{in}_G(v)}_{1,\ldots,\mathsf{arity}(v)}]$, even conditioned on the values of the fewer than $\lambda^k \leq b$ queries made by $\tilde{V}$ to R_v. Therefore any string $\big(\dot{v}[\mathtt{x}](\vec{\gamma}_1), \ldots, \dot{v}[\mathtt{x}](\vec{\gamma}_\ell)\big)$ for $\ell \leq \mathsf{in}_G(v)$ and distinct $\vec{\gamma}_1, \ldots, \vec{\gamma}_\ell \in (\mathbb{F} - H)^{\mathsf{arity}(v)}$ is identically distributed to a uniformly random string in $\mathbb{F}^\ell$. Observe that for every internal vertex v in V, $|L_v| \leq \mathsf{in}_G(v)$, and the prover sends $\dot{v}[\mathtt{x}](\vec{\gamma})$ for each $(\vec{\gamma}, a) \in L_v$, so all of these values are uniformly random in $\mathbb{F}$, and thus identically distributed to the values h^i_{sim} the simulator sends.

Clearly R^v_{sim} and R_v are identically distributed for every internal vertex v in V. The perfect zero knowledge property of the strong ZK sumcheck simulator guarantees that the simulation of π_v and the messages sent during the sumcheck protocol is perfect given a single query to the oracle F. We simulate this query by substituting values h^i_{sim} in place of $\dot{u}_i[\mathtt{x}](\vec{c}_i)$, which we argue above are identically distributed.

It follows from the description and the efficiency of the subsimulator that the simulator runs in time $\mathrm{poly}(|\mathcal{C}|) \cdot |\mathbb{x}| + |V(G)|(\mathsf{arity}(G)+k)(\lambda \mathsf{q}_{\vec{V}}|H| + \lambda^3 \mathsf{q}_{\vec{V}}^3) \cdot \mathrm{poly}(\log|\mathbb{F}|)$, provided we use the algorithm of Corollary 4.1 for Step 1. □

9.2 The case of sum-product satisfaction

To make the sum-product circuit satisfaction protocol zero knowledge, in addition to the above considerations we must also avoid leaking information about the witness $\mathbb{x}$. This we achieve using similar techniques to those seen previously: for each leaf vertex w, rather than directly sending the auxiliary input polynomial $\mathbb{z}_w(\vec{X})$, the prover sends a polynomial $\mathbb{z}'_w(\vec{X},\vec{Y})$ chosen randomly from the set of low-degree polynomials summing over H^k to $\mathbb{z}_w(\vec{X})$ for some appropriately-chosen k. This acts as a perfectly hiding commitment to the witness.

We show that we can efficiently construct a sum-product circuit $\mathcal{C}'$ which outputs $\mathbb{y}$ on input $(\mathbb{x}, \mathbb{z}'_w)$ if and only if the original sum-product circuit outputs $\mathbb{y}$ on input $(\mathbb{x}, \mathbb{z}_w)$. We then obtain our zero-knowledge protocol by running the protocol of Theorem 9.1 on $\mathcal{C}'$, implementing the queries to the input as queries to the proof (with self-correction).

Theorem 9.2 (PZK IPCP for $\mathscr{R}_{\mathrm{SPCS}}$)**.** *For every query bound function $b(n)$, the relation $\mathscr{R}_{\mathrm{SPCS}}$ has a (public-coin and non-adaptive) Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail, letting $\alpha := \log b / \log|H|$:*

$$\mathscr{R}_{\mathrm{SPCS}} \in \mathbf{PZK\text{-}IPCP}\left[\begin{array}{rl} \textit{soundness error:} & O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{in}(G) \cdot (\mathsf{arity}(G)+\alpha) \cdot |V(G)|/|\mathbb{F}|) \\ \textit{round complexity:} & O(\mathsf{depth}(G) \cdot (\mathsf{arity}(G)+\alpha)) \\ \textit{proof length:} & O(|V(G)| \cdot |\mathbb{F}|^{\mathsf{arity}(G)+2\alpha}) \\ \textit{query complexity:} & |V(G)| \cdot \mathrm{poly}(\log|\mathbb{F}|, \mathsf{arity}(G), \alpha, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, \mathsf{in}(G)) \\ \textit{prover time:} & \mathrm{poly}(|\mathcal{C}|, |\mathbb{x}|, |\mathbb{z}|, |\mathbb{F}|^{\mathsf{arity}(G)+\alpha}) \\ \textit{verifier time:} & \mathrm{poly}(|\mathcal{C}|, |\mathbb{x}|) \\ \textit{verifier space:} & O((\mathsf{arity}(G)+\alpha) \cdot \mathsf{width}(G) \cdot \mathsf{in}(G) \cdot \log|\mathbb{F}| + \log|\mathcal{C}| + \mathsf{space}(\mathbb{x})) \\ \textit{simulator overhead:} & \mathrm{poly}(|\mathcal{C}|, \alpha) \cdot (|\mathbb{x}| + \mathsf{q}_{\vec{V}}^3). \end{array}\right].$$

Proof. Fix $k := \log b / \log|H|$. Let $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ be a sum-product circuit and let $G = (V, E, \rho, \sigma)$ be its ari-graph. Let $\mathbb{x}$ be a partial mapping of the leaf vertices of G to arithmetic circuits. We construct the new sum-product circuit $\mathcal{C}' = (\mathbb{F}, H, \delta_{\mathsf{in}}, \max\{\delta_{\mathsf{lf}}, 2|H|\}, G', C')$ where:

- the new ari-graph $G' = (V', E', \rho', \sigma')$ extends G by adding a new vertex v_w and a new edge (w, v_w) for each leaf vertex w not in the domain of $\mathbb{x}$ as follows

$$\begin{aligned} V' &:= V \cup \{v_w : w \in V \text{ is a leaf}\}\ , \\ E' &:= E \cup \{(w, v_w) : w \in V \text{ is a leaf}\}\ , \\ \rho'_e &:= \begin{cases} \rho_e & \text{if } e \in E \\ \{1, \ldots, \mathsf{arity}(w)\} & \text{if } e = (w, v_w) \text{ and } w \text{ is a leaf in } V \end{cases}\ , \\ \sigma'_e &:= \begin{cases} \sigma_e & \text{if } e \in E \\ \{1, \ldots, k\} & \text{if } e = (w, v_w) \text{ and } w \text{ is a leaf in } V \end{cases}\ . \end{aligned}$$

  Note that the vertex and edge sets at most double in size, $\mathsf{depth}(G') = \mathsf{depth}(G) + 1$, $\mathsf{arity}(v_w) = \mathsf{arity}(w) + k$ and $m_{v_w} = k$.

- C' is such that $C'_v := C_v$ for every internal vertex v in V, and C'_v is the univariate polynomial X for every leaf vertex v in V.

Before describing the protocol, we prove a claim that relates the two sum-product circuits $\mathcal{C}$ and $\mathcal{C}'$.

Claim. *For every $\mathbb{y} \in \mathbb{F}$, $(\mathcal{C}, \mathbb{y}, \mathbb{x}) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$ if and only if $(\mathcal{C}', \mathbb{y}, (\mathbb{x}, \mathbb{z}')) \in \mathscr{L}_{\mathrm{SPCE}}$ for some input $\mathbb{z}'$ for $\mathcal{C}'$ such that $\mathbb{z}'_{v_w} \in \mathbb{F}[X_{1,\ldots,\mathsf{arity}(w)}^{\leq \delta_{\mathsf{lf}}}, Y_{1,\ldots,k}^{\leq 2|H|}]$ for every leaf w in V.*

Proof of claim. First suppose that $(\mathcal{C}, \mathtt{y}, \mathtt{x}) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$, so there exists an auxiliary input $\mathtt{z}$ for $\mathcal{C}$ such that $\mathcal{C}[\mathtt{x}, \mathtt{z}] = \mathtt{y}$. Define $\mathtt{z}'$ to be the auxiliary input for $\mathcal{C}'$ such that $\mathtt{z}'_{v_w}(\vec{X}, \vec{Y}) := I_{H^k}(\vec{Y}, \vec{0}) \cdot \mathtt{x}_w(\vec{X})$ for every leaf w in V in the domain of $\mathtt{z}$. Note that for each such leaf w in V, $\mathtt{z}'_{v_w}$ respects the desired degree bounds and, moreover, $w[\mathtt{x}, \mathtt{z}'](\vec{X}) = \sum_{\vec{\beta} \in H^k} I_{H^k}(\vec{\beta}, \vec{0}) \cdot \mathtt{z}_w(\vec{X}) = \mathtt{z}_w(\vec{X}) = w[\mathtt{x}, \mathtt{z}](\vec{X})$. By construction of G', we deduce that $\mathcal{C}'[\mathtt{x}, \mathtt{z}'] = \mathtt{y}$, so that $(\mathcal{C}', \mathtt{y}, (\mathtt{x}, \mathtt{z}')) \in \mathscr{L}_{\mathrm{SPCE}}$.

Next suppose that $(\mathcal{C}', \mathtt{y}, (\mathtt{x}, \mathtt{z}')) \in \mathscr{L}_{\mathrm{SPCE}}$ for some input $\mathtt{z}'$ for $\mathcal{C}'$ such that $\mathtt{z}'_{v_w} \in \mathbb{F}[X^{\leq \delta_{\mathsf{lf}}}_{1,\ldots,\mathsf{arity}(w)}, Y^{\leq 2|H|}_{1,\ldots,k}]$ for every leaf w in V not in the domain of $\mathtt{x}$. Define $\mathtt{z}$ to be the input for $\mathcal{C}$ such that $\mathtt{z}_w(\vec{X}) := \sum_{\beta \in H^k} \mathtt{z}'_{v_w}(\vec{X}, \vec{\beta})$ for each leaf w in V. Note that for each leaf w in V not in the domain of $\mathtt{x}$, $\mathtt{z}_w$ has individual degree at most δ_{lf} and, moreover, $w[\mathtt{x}, \mathtt{z}](\vec{X}) = \sum_{\beta \in H^k} \mathtt{z}'_{v_w}(\vec{X}, \vec{\beta}) = w[\mathtt{x}, \mathtt{z}'](\vec{X})$. By construction of G', this implies that $\mathcal{C}[\mathtt{x}, \mathtt{z}'] = \mathtt{y}$, so that $(\mathcal{C}, \mathtt{y}, \mathtt{x}) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$. □

The protocol proceeds as follows. The prover and verifier receive a SPCS instance $(\mathcal{C}, \mathtt{y}, \mathtt{x})$ as input, and the prover additionally receives an auxiliary input $\mathtt{z}$ for $\mathcal{C}$ that is a valid witness for $(\mathcal{C}, \mathtt{y}, \mathtt{x})$. Both use $\mathcal{C}$ to construct the new sum-product circuit $\mathcal{C}'$ from the above claim; in addition, the prover uses $\mathtt{z}$ to sample an input $\mathtt{z}'$ for $\mathcal{C}'$ by choosing, for each leaf w in the domain of $\mathtt{z}$, a polynomial $\mathtt{z}'_{v_w} \in \mathbb{F}[X^{\leq \delta_{\mathsf{lf}}}_{1,\ldots,\mathsf{arity}(w)}, Y^{\leq 2|H|}_{1,\ldots,k}]$ uniformly at random conditioned on $\sum_{\vec{\beta} \in H^k} \mathtt{z}'_{v_w}(\vec{\alpha}, \vec{\beta}) = \mathtt{z}_w(\vec{\alpha})$ for all $\vec{\alpha} \in \mathbb{F}^{\mathsf{arity}(w)}$. The prover and verifier then engage in the zero knowledge Interactive Probabilistically Checkable Proof for the language $\mathscr{L}_{\mathrm{SPCE}}$ (see Theorem 9.1) on input $(\mathcal{C}', \mathtt{y}, (\mathtt{x}, \mathtt{z}'))$, with the prover appending the auxiliary input $\mathtt{z}'$ to the proof oracle. At the end of the protocol the verifier needs to make a single query $\vec{\gamma}_{v_w}$ to $\mathtt{z}'_{v_w}$ for each leaf vertex w in the domain of $\mathtt{x}$, so the verifier tests that $\mathtt{z}'_{v_w}$ is close to the evaluation of a polynomial in $\mathbb{F}[X^{\leq \delta_{\mathsf{lf}}}_{1,\ldots,\mathsf{arity}(w)}, Y^{\leq 2|H|}_{1,\ldots,k}]$ and then uses self-correction to read $\mathtt{z}'_{v_w}(\vec{\gamma}_{v_w})$.

Completeness is straightforward to argue; we only discuss soundness and then zero knowledge.

Suppose that $(\mathcal{C}, \mathtt{y}, \mathtt{x}) \notin \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$, and let $\tilde{\mathtt{z}}$ be the input for $\mathcal{C}'$ that the prover has appended to the proof oracle. If there exists a leaf vertex w in the domain of $\tilde{\mathtt{z}}$ such that $\tilde{\mathtt{z}}_{v_w}$ is more than δ-far from a polynomial in $\mathbb{F}[X^{\leq \delta_{\mathsf{lf}}}_{1,\ldots,\mathsf{arity}(w)}, Y^{\leq 2|H|}_{1,\ldots,k}]$, then the verifier accepts with probability at most ϵ. So suppose that this is not the case, and let $\tilde{\mathtt{z}}'_{v_w}$ be the unique polynomial in $\mathbb{F}[X^{\leq \delta_{\mathsf{lf}}}_{1,\ldots,\mathsf{arity}(w)}, Y^{\leq 2|H|}_{1,\ldots,k}]$ that is δ-close to $\tilde{\mathtt{z}}_{v_w}$. Using self-correction, the verifier obtains $\tilde{\mathtt{z}}'_{v_w}(\vec{\gamma}_{v_w})$ with probability at least $1 - \epsilon$. By a union bound, the probability that the verifier learns all the values correctly is at least $1 - \epsilon|V(G)|$. By the claim, $(\mathcal{C}, \mathtt{y}, \mathtt{x}) \notin \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$ implies that $(\mathcal{C}', \mathtt{y}, (\mathtt{x}, \tilde{\mathtt{z}}')) \notin \mathscr{L}_{\mathrm{SPCE}}$, so if all of the verifier's queries are answered according to $\tilde{\mathtt{z}}'$, the soundness of the protocol for $\mathscr{L}_{\mathrm{SPCE}}$ implies that the verifier accepts with probability at most $O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{in}(G) \cdot (\mathsf{arity}(G) + \log b) \cdot |V(G)|/|\mathbb{F}|)$. Setting ϵ and δ appropriately yields the claimed soundness error.

Perfect zero knowledge follows from Corollary 5.3: if a verifier makes fewer than $|H|^k$ queries to the proof, then each $\mathtt{z}'_v$ is indistinguishable from a random polynomial. The simulator for this protocol simply runs the simulator from Theorem 9.1 on $(\mathcal{C}', \mathtt{y}, (\mathtt{x}, \mathtt{z}'))$, and uses the algorithm of Corollary 4.1 to simulate its queries to each $\mathtt{z}'_v$. □

10 Zero knowledge for polynomial space

We present an efficient reduction from problems decidable in polynomial space (**PSPACE**) to sum-product circuit evaluation problems ($\mathscr{L}_{\mathrm{SPCE}}$, Definition 8.9). The reduction yields perfect zero knowledge IPCPs for **PSPACE**, via our construction of perfect zero knowledge IPCPs for $\mathscr{L}_{\mathrm{SPCE}}$ (Theorem 9.1), resolving an open problem of [BCFGRS16].

Our starting point is the language of true quantified boolean formulas (TQBFs), which is **PSPACE**-complete:

Definition 10.1. *Let $\mathscr{L}_{\mathrm{TQBF}}$ be the language of quantified boolean formulas $\Phi = \mathrm{Q}_1 x_1 \cdots \mathrm{Q}_n x_n \phi(x_1, \ldots, x_n)$, with $\mathrm{Q}_i \in \{\forall, \exists\}$, that evaluate to true. We denote by n the number of variables and by c the number of clauses in ϕ.*

The theorem below is a 'zero knowledge analogue' of Shamir's protocol for $\mathscr{L}_{\mathrm{TQBF}}$ [Sha92], sharing all its key features except that it is an Interactive PCP rather than an Interactive Proof. We prove that $\mathscr{L}_{\mathrm{TQBF}}$ has a public-coin Interactive PCP that is perfect zero knowledge, with exponential proof length and polynomial query complexity; also, like Shamir's protocol, the number of rounds is $O(n^2)$, the prover runs in space $\mathrm{poly}(c)$ and the verifier in time $\mathrm{poly}(c)$.

Theorem 10.2 (PZK IPCP for **PSPACE**)**.** *For every query bound function $\mathsf{b}(n)$, the **PSPACE**-complete language $\mathscr{L}_{\mathrm{TQBF}}$ has a (public-coin and non-adaptive) Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail:*

$$\mathscr{L}_{\mathrm{TQBF}} \in \mathbf{PZK\text{-}IPCP} \left[\begin{array}{rl} \textit{soundness error:} & 1/2 \\ \textit{round complexity:} & O(n \cdot (n + \log \mathsf{b})) \\ \textit{proof length:} & (|\Phi| + \log \mathsf{b})^{O(n + \log \mathsf{b})} \\ \textit{query complexity:} & \mathrm{poly}(|\Phi|, \log \mathsf{b}) \\ \textit{prover time:} & (|\Phi| + \log \mathsf{b})^{O(n + \log \mathsf{b})} \\ \textit{verifier time:} & \mathrm{poly}(n, \log \mathsf{b}) + O(c) \\ \textit{verifier space:} & O((n + \log \mathsf{b}) \cdot \log |\Phi|) \\ \textit{simulator overhead:} & \mathrm{poly}(n + \log \mathsf{b}) \cdot \mathsf{q}_V^3. \end{array} \right] .$$

We first state and prove the reduction from $\mathscr{L}_{\mathrm{TQBF}}$ to $\mathscr{L}_{\mathrm{SPCE}}$ (Lemma 10.3 below), and then prove the theorem. The reduction is based on Shen's arithmetization of QBFs [She92] and (at least implicitly) his idea of degree reduction. (In particular, we do not rely on Shamir's restriction to 'simple' QBFs [Sha92].) We use Shen's arithmetization in order to ensure that the arithmetized expression only takes boolean values.

Lemma 10.3 ($\mathscr{L}_{\mathrm{TQBF}} \to \mathscr{L}_{\mathrm{SPCE}}$)**.** *There exists a polynomial-time function f such that, for every quantified boolean formula Φ and prime p, $f(\Phi, p) \in \mathscr{L}_{\mathrm{SPCE}}$ if and only if Φ is true. Moreover, if Φ has n variables and c clauses then $f(\Phi, p)$ is a sum-product circuit instance $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ with $|\mathbb{F}| = p$, $|H| = \Theta(1)$, $\delta_{\mathsf{in}} = \Theta(1)$, $\delta_{\mathsf{lf}} = \Theta(c)$, $|V(G)| = \Theta(n)$, $\mathsf{arity}(G) = \Theta(n)$, $\mathsf{in}(G) = \Theta(1)$, $\mathsf{width}(G) = \Theta(1)$, $\mathsf{space}(\mathtt{x}) = O(\log |\Phi|)$.*

Proof. Let Φ be a quantified boolean formula that, without loss of generality, has the following 'regular' form:

$$\Phi = \forall x_1 \exists x_2 \cdots \forall x_{n-1} \exists x_n \, \phi(x_1, x_2, \ldots, x_{n-1}, x_n) \ ,$$

where ϕ is a 3-CNF formula with c clauses, and n is even. This regular form is achievable with only constant multiplicative overheads in the number of variables and clauses. We arithmetize the formula and quantifiers.

- The arithmetization of a 3-CNF formula ϕ with variables $z_1, \ldots, z_n$ and clauses $K_1, \ldots, K_c$ is the polynomial $\hat{\phi}$ of total degree at most $3c$ given by: $\hat{\phi}(Z_1, \ldots, Z_n) := \prod_{K \in \phi}(1 - \prod_{\{i : z_i \in K\}}(1 - Z_i) \cdot \prod_{\{i : \bar{z}_i \in K\}} Z_i)$. Note that $\hat{\phi}(x_1, \ldots, x_n) = \phi(x_1, \ldots, x_n)$ for all boolean $x_1, \ldots, x_n$ (i.e., they agree on the boolean hypercube).
- The arithmetization of the two quantifiers is as follows: $\forall x \, \phi(x)$ maps to $\prod_{x \in \{0,1\}} \hat{\phi}(x)$ and $\exists x \, \phi(x)$ maps to $\coprod_{x \in \{0,1\}} \hat{\phi}(x) := \big(1 - (1 - \hat{\phi}(0))(1 - \hat{\phi}(1))\big)$. These arithmetic expressions have the same value as the boolean expressions (over any field); in particular they are 0 or 1.

For any prime p, consider the following construction.

- Use n, c, p to construct a sum-product circuit instance $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ where

$$\mathbb{F} := \mathbb{F}_p\,;\ H := \{0,1\}\,;\ \delta_{\mathsf{in}} := 4\,;\ \delta_{\mathsf{lf}} := 3c\,;$$

the ari-graph $G := (V, E)$ is defined as follows

$$\begin{aligned}
V &:= \{v_i\}_{i\in\{0,\ldots,n\}} \\
E &:= \{e_i, e_i' = (v_i, v_{i+1})\}_{i\in\{0,\ldots,n-1\}} \\
\rho_e &:= \{1,\ldots,i\} \quad \text{if } e = e_i \text{ or } e = e_i' \text{ for some } i \in \{0,\ldots,n-1\}, \\
\sigma_e &:= \begin{cases} \{1\} & \text{if } e = e_i \text{ for some } i \in \{0,\ldots,n-1\} \text{ or} \\ \{2\} & \text{if } e = e_i' \text{ for some } i \in \{0,\ldots,n-1\}, \end{cases} \\
C_{v_i} &:= \begin{cases} (1-X_i)X_i \cdot ZZ' & \text{if } i \in \{0,\ldots,n-1\} \text{ is even or} \\ (1-X_i)X_i \cdot \big(1-(1-Z)(1-Z')\big) & \text{if } i \in \{0,\ldots,n-1\} \text{ is odd,} \end{cases}
\end{aligned}$$

where Z, Z' in both cases correspond to the edges e_i, e_i'.

- Construct an input $\mathtt{x}$ for $\mathcal{C}$ that maps v_n to the polynomial $\hat{\phi}$ in $\mathbb{F}_p[X_{1,\ldots,n}^{\leq 3c}]$ that equals the arithmetization of ϕ.

See Figure 1, Figure 2, Figure 3 for diagrams of this sum-product circuit and input for 2, 4, n variables respectively.

We claim that for every even i in $\{0,\ldots,n\}$ and $x_1,\ldots,x_i \in \{0,1\}$:

$$v_i[\mathtt{x}](x_1, x_2 \ldots, x_{i-1}, x_i) = \prod_{x_{i+1}\in\{0,1\}} \coprod_{x_{i+2}\in\{0,1\}} \cdots \prod_{x_{n-1}\in\{0,1\}} \coprod_{x_n\in\{0,1\}} \hat{\phi}(x_1, x_2, \ldots, x_{n-1}, x_n)\ . \tag{13}$$

We argue the equality by induction on i. When $i = n$, Equation 13 is $v_n[\mathtt{x}] = \hat{\phi}$, which holds by definition; next, we assume the equality for $i+2$ and prove it for i. By construction, for every $x_1,\ldots,x_i \in \{0,1\}$,

$$\begin{aligned}
v_{i+1}[\mathtt{x}](x_1,\ldots,x_i) &= \sum_{x_{i+1},x_{i+1}'\in\{0,1\}} (1-x_{i+1})x_{i+1}' \cdot (1-(1-v_{x_{i+2}}[\mathtt{x}](x_1,\ldots,x_i,x_{i+1}))(1-v_{x_{i+2}}[\mathtt{x}](x_1,\ldots,x_i,x_{i+1}'))) \\
&= (1-(1-v_{x_{i+2}}[\mathtt{x}](x_1,\ldots,x_i,0))(1-v_{x_{i+2}}[\mathtt{x}](x_1,\ldots,x_i,1))) \\
&= \coprod_{x_{i+2}\in\{0,1\}} \cdots \prod_{x_{n-1}\in\{0,1\}} \coprod_{x_n\in\{0,1\}} \hat{\phi}(x_1, x_2, \ldots, x_{n-1}, x_n)\ ,
\end{aligned}$$

where the second equality follows by the inductive assumption. Then, by a similar argument,

$$\begin{aligned}
v_i[\mathtt{x}](x_1,\ldots,x_i) &= v_{i+1}[\mathtt{x}](x_1,\ldots,x_i,0)\cdot v_i[\mathtt{x}](x_1,\ldots,x_i,1) \\
&= \prod_{x_{i+1}\in\{0,1\}} \coprod_{x_{i+2}\in\{0,1\}} \cdots \prod_{x_{n-1}\in\{0,1\}} \coprod_{x_n\in\{0,1\}} \hat{\phi}(x_1, x_2, \ldots, x_{n-1}, x_n)\ ,
\end{aligned}$$

which completes the induction.

We now turn to the reduction's completeness and soundness. The key fact, which immediately follows from Equation 13 and the arithmetization's properties, is that the quantified boolean formula Φ is true if and only if $\mathcal{C}[\mathtt{x}] = v_{x_1}[\mathtt{x}]$ is 1 (over any field $\mathbb{F}$). Thus, regardless of the choice of prime p, Φ is true if and only if $(\mathcal{C}, 1, \mathtt{x}) \in \mathscr{L}_{\mathrm{SPCE}}$. □

Proof of Theorem 10.2. The prover and verifier receive as input a quantified boolean formula Φ. They agree on a deterministic procedure to find a prime in the range $[cn^3 \log \mathsf{b}, 2cn^3 \log \mathsf{b}]$; this can easily be done in polynomial time by exhaustive search and brute-force primality testing. The prover and the verifier then engage in the perfect zero knowledge Interactive PCP for $\mathscr{L}_{\mathrm{SPCE}}$ (Theorem 9.1) on the input $f(\Phi, p)$. The soundness error of the protocol is $O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{in}(G) \cdot (\mathsf{arity}(G) + \log \mathsf{b}) \cdot |V(G)|/|\mathbb{F}|) = O(cn^2 \log \mathsf{b}/p)$; the size of p ensures that the soundness error is $O(1/n) < 1/2$ for sufficiently large n. □

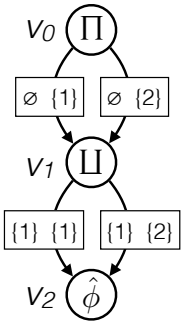


Figure 1: Sum-product circuit and its input for QBF with 2 variables.

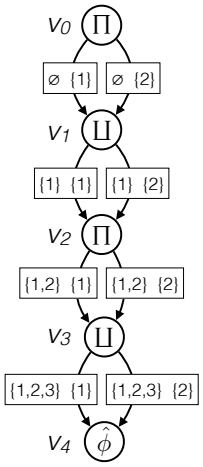


Figure 2: Sum-product circuit and its input for QBF with 4 variables.

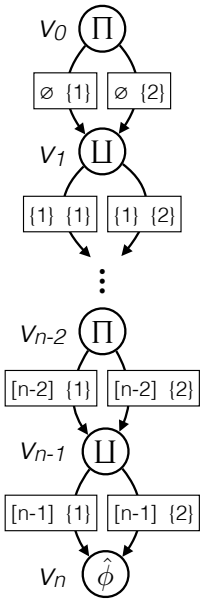


Figure 3: Sum-product circuit and its input for QBF with n variables.

11 Zero knowledge for the evaluation of low-depth circuits

We present a 'zero knowledge analogue' of GKR's protocol for circuit evaluation [GKR15], except that the protocol is an Interactive PCP rather than an Interactive Proof. We achieve this by exhibiting an efficient reduction from circuit evaluation (for certain circuits) to sum-product circuit evaluation problems ($\mathscr{L}_{\mathrm{SPCE}}$, Definition 8.9). The reduction then yields the desired zero knowledge protocol (Theorem 11.1 below), via our construction of perfect zero knowledge Interactive PCPs for $\mathscr{L}_{\mathrm{SPCE}}$ (Theorem 9.1). We now provide context, state the theorem, and then describe its proof.

Goldwasser, Kalai, and Rothblum [GKR15] give *interactive proofs for muggles* (also known as *doubly-efficient interactive proofs* [RRR16]) for low-depth circuits that are sufficiently uniform.

Namely, given a language $\mathscr{L}$ decidable by a family of $O(\log S(n))$-space uniform boolean circuits of size $S(n)$ and depth $D(n)$, [GKR15] gives a public-coin Interactive Proof for $\mathscr{L}$ where the prover runs in time $\mathrm{poly}(S(n))$ and the verifier runs in time $(n + D(n)) \cdot \mathrm{poly}(\log S(n))$ and space $O(\log S(n))$; the number of rounds and communication complexity are $D(n) \cdot \mathrm{poly}(\log S(n))$. Moreover, if the verifier is given oracle access to the low-degree extension of the circuit's input, the verifier runs only in time $D(n) \cdot \mathrm{poly}(\log S(n))$; this *sub-linear* running time enables applications to Interactive Proofs *of Proximity* [RVW13].

The theorem below provides similar features, but in addition also provides perfect zero knowledge, via an Interactive PCP instead of an Interactive Proof.

Theorem 11.1 (PZK IPCP for low-depth uniform boolean circuits)**.** *Let $\mathscr{L}$ be a language decidable by a family of $O(\log S(n))$-space uniform boolean circuits of size $S(n)$ and depth $D(n)$, and let $b(n)$ be a query bound function. Then $\mathscr{L}$ has a (public-coin and non-adaptive) Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail:*

$$\mathscr{L} \in \mathbf{PZK\text{-}IPCP} \left[\begin{array}{rl} \textit{soundness error:} & 1/2 \\ \textit{round complexity:} & D(n) \cdot \log b(n) \cdot \mathrm{poly}(\log S(n)) \\ \textit{proof length:} & D(n) \cdot \mathrm{poly}(S(n), b(n)) \\ \textit{query complexity:} & D(n) \cdot \mathrm{poly}(\log S(n), \log b(n)) \\ \textit{prover time:} & D(n) \cdot \mathrm{poly}(S(n), b(n)) \\ \textit{verifier time:} & n \cdot \mathrm{poly}(D(n), \log S(n), \log b(n)) \\ \textit{verifier space:} & O(\log S(n) + \log b(n) / \log\log S(n)) \\ \textit{simulator overhead:} & \mathrm{poly}(D(n), \log S(n), \log b(n)) \cdot (n + \mathsf{q}_V^3). \end{array} \right].$$

Moreover, if the verifier (resp. simulator) is given oracle access to the low-degree extension of the circuit's input, the verifier runs in time $D(n) \cdot \mathrm{poly}(\log S(n), \log b(n))$, and the simulator overhead is $\mathrm{poly}(D(n), \log S(n), \log b(n)) \cdot \mathsf{q}_V^3$.

Our proof of the theorem follows the structure in [GKR15]. After recalling some notions for circuits (Section 11.1) and extending some of our definitions (Section 11.2), we proceed in three steps:

- *Step 1* (Section 11.3). We reduce the evaluation of a given layered arithmetic circuit to the evaluation of a sum-product circuit, when given oracles for low-degree extensions for the circuit's wiring predicates. One can view this step as casting the *barebones protocol* of [GKR15, Section 3] in the framework of sum-product circuits.
- *Step 2* (Section 11.4). We reduce small-space Turing machine computations to sum-product circuit evaluations.
- *Step 3* (Section 11.5). We combine the previous two steps to prove the theorem, using the fact that small-space Turing machines can evaluate low-degree extensions of the wiring predicates for sufficiently uniform boolean circuits.

Before discussing each of the above steps, we discuss consequences for Turing machine computations.

As in [GKR15], we can derive from the above theorem a useful corollary for Turing machine computations, using their reduction from Turing machines to boolean circuits.

Corollary 11.2 (PZK IPCP for Turing machines)**.** *Let $\mathscr{L}$ be a language decidable by a (deterministic) Turing machine in space $s(n) = \Omega(\log n)$, and let $b(n)$ be a query bound function. Then $\mathscr{L}$ has a (public-coin and non-adaptive)*

Interactive PCP that is perfect zero knowledge against all b-query malicious verifiers. In more detail:

$$\mathscr{L} \in \mathbf{PZK\text{-}IPCP}\left[\begin{array}{rl} \textit{soundness error:} & 1/2 \\ \textit{round complexity:} & \mathrm{poly}(s(n)) \cdot \log b(n) \\ \textit{proof length:} & \mathrm{poly}(2^{s(n)}, b(n)) \\ \textit{query complexity:} & \mathrm{poly}(s(n), b(n)) \\ \textit{prover time:} & \mathrm{poly}(2^{s(n)}, b(n)) \\ \textit{verifier time:} & n \cdot \mathrm{poly}(s(n), \log b(n)) \\ \textit{verifier space:} & O(s(n) + \log b(n) / \log s(n)) \end{array}\right] .$$

Proof. By [GKR15, Lemma 4.1], a language $\mathscr{L}$ decidable by a Turing machine in time $t(n)$ and space $s(n)$ is also decidable by a $O(s(n))$-space uniform circuit family of size $\mathrm{poly}(t(n)2^{s(n)}) = \mathrm{poly}(2^{s(n)})$ and depth $\mathrm{poly}(s(n))$. Applying Theorem 11.1 to this circuit family yields the corollary. □

In each of the results above, one can set the query bound $b(n)$ to be superpolynomial in n (say, $b(n) := n^{O(\log\log n)}$) to obtain Interactive PCPs that are perfect zero knowledge against all polynomial-time malicious verifiers, without affecting the efficiency of the honest verifier.

11.1 Notations for layered arithmetic circuits

We briefly recall some definitions and observations from [GKR15].

Definition 11.3. *A* layered arithmetic circuit $C\colon \mathbb{F}^n \to \mathbb{F}$ *of depth D and size S (with $n \le S$) is an arithmetic circuit, with fan-in 2, arranged into $D+1$ layers: the output layer (layer 0) has a single gate; layers $1, \ldots, D-1$ have S gates each; and the input layer (layer D) has n gates. For $i \in \{0, \ldots, D-1\}$, each gate in layer i has two inputs, which are gates in layer $i+1$; the n gates in layer D are C's inputs; the single gate in layer 0 is C's output.*

For $i \in \{1, \ldots, D-1\}$, we denote by $V_i\colon \mathbb{F}^n \times [S] \to \mathbb{F}$ the function such that $V_i(\vec{x}, j) = v$ if and only if the j-th gate of the i-th layer has value v when C's input is $\vec{x} \in \mathbb{F}^n$. Moreover, we denote by $V_D\colon \mathbb{F}^n \times [n] \to \mathbb{F}$ the function $V_D(\vec{x}, j) := x_j$, and by $V_D\colon \mathbb{F}^n \to \mathbb{F}$ the function $V_0(\vec{x}) := C(x)$.

Let $H \subseteq \mathbb{F}$ and $m, m' \in \mathbb{N}$ be such that $|H| \ge 2$, $m \ge \lceil \log S / \log |H| \rceil$, and $m' \ge \lceil \log n / \log |H| \rceil$.

For $i \in \{1, \ldots, D-1\}$, we can equivalently view V_i as a function from $\mathbb{F}^n \times H^m$ to $\mathbb{F}$, by taking an arbitrary order α on H^m, and letting $V_i(\vec{x}, \vec{z}) = 0$ for every $\vec{z} \in H^m$ with $\alpha(z) > S$. The following equation then relates V_{i-1} to V_i:

$$V_{i-1}(\vec{x}, \vec{z}) = \sum_{\vec{\omega}_1, \vec{\omega}_2 \in H^m} \mathrm{add}_i(\vec{z}, \vec{\omega}_1, \vec{\omega}_2) \cdot \big(V_i(\vec{x}, \vec{\omega}_1) + V_i(\vec{x}, \vec{\omega}_2)\big) + \mathrm{mul}_i(\vec{z}, \vec{\omega}_1, \vec{\omega}_2) \cdot \big(V_i(\vec{x}, \vec{\omega}_1) \cdot V_i(\vec{x}, \vec{\omega}_2)\big)$$

where $\mathrm{add}_i\colon H^{3m} \to \mathbb{F}$ (resp., $\mathrm{mul}_i\colon H^{3m} \to \mathbb{F}$) is the predicate such that, for every $(\vec{a}, \vec{b}, \vec{c}) \in H^{3m}$, $\mathrm{add}_i(\vec{a}, \vec{b}, \vec{c})$ (resp., $\mathrm{mul}_i(\vec{a}, \vec{b}, \vec{c})$) equals 1 if the $\alpha(\vec{a})$-th gate of layer $i-1$ is an addition (resp., multiplication) gate whose inputs are gates $\alpha(\vec{b}) \le \alpha(\vec{c})$ of layer i, or 0 otherwise. The situation for the input layer and output layer is somewhat different.

- The input layer (layer D) has n gates (rather than S), so we can equivalently view V_D as a function from $\mathbb{F}^n \times H^{m'}$ to $\mathbb{F}$, by taking an arbitrary order α' on $H^{m'}$, and letting $V_D(\vec{x}, \vec{z}) = 0$ for every $\vec{z} \in H^{m'}$ with $\alpha'(z) > n$. Naturally, we need to adjust the expression for V_{D-1} and the definitions of $\mathrm{add}_D, \mathrm{mul}_D$ appropriately.
- The output layer (layer 0) has a single gate (rather than S), so and we can write

$$V_0(\vec{x}) = \sum_{\vec{\omega}_1, \vec{\omega}_2 \in H^m} \mathrm{add}_1(\vec{0}, \vec{\omega}_1, \vec{\omega}_2) \cdot \big(V_1(\vec{x}, \vec{\omega}_1) + V_1(\vec{x}, \vec{\omega}_2)\big) + \mathrm{mul}_1(\vec{0}, \vec{\omega}_1, \vec{\omega}_2) \cdot \big(V_1(\vec{x}, \vec{\omega}_1) \cdot V_1(\vec{x}, \vec{\omega}_2)\big) \ .$$

Remark 11.4. For $i \in \{1, \ldots, D\}$, we view V_i as a function not only of the gate number j (represented as $\vec{z} \in H^m$) but also of the circuit's input $\vec{x}$, and we view V_0 as a function of the input only. In contrast, [GKR15] defines $V_0, \ldots, V_D$ with $\vec{x}$ 'hard-coded'. We require the additional flexibility ($\vec{x}$ is an input) to compose sum-product circuits below.

11.2 Sum-product subcircuits and oracle inputs

Later on we will need to assemble different sum-product circuits into one such circuit, which requires relaxing the definition of an ari-graph to allow the root to have positive arity. This is stated formally below (difference highlighted).

Definition 11.5 (extends Definition 8.7)**.** *A tuple $G = (V, E, \rho, \sigma)$ is an* **ari-graph** *if (V, E) is a directed acyclic multi-graph and both ρ and σ label every edge e in E with finite sets of positive integers ρ_e and σ_e that satisfy the following property. For every vertex v in V, there exists a (unique) non-negative integer $\mathsf{arity}(v)$ such that, if $e_1, \ldots, e_t$ are v's outgoing edges: (1) if v is the root then $\mathsf{arity}(v) = \max(\sigma_{e_1} \cup \ldots \cup \sigma_{e_t})$, otherwise $\mathsf{arity}(v) = |\rho_{e_1}| + |\sigma_{e_1}| = \cdots = |\rho_{e_t}| + |\sigma_{e_t}|$ where $e_1, \ldots, e_{\mathsf{in}_G(v)}$ are v's incoming edges; (2) $\rho_{e_1}, \ldots, \rho_{e_t} \subseteq \{1, \ldots, \mathsf{arity}(v)\}$.*

For consistency, we retain the original definition of sum-product circuits, where the root must have arity zero, and so its value is a constant in $\mathbb{F}$. Instead, we now define sum-product *sub*circuits, where the root may have positive arity.

Definition 11.6. *A* **sum-product subcircuit** *$\mathcal{C}$ is a tuple $(\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ whose definition is identical to that of a sum-product circuit (see Definition 8.8), except that the root of G may have positive arity. In particular, $\mathcal{C}[\mathtt{x}]$ may be a non-constant polynomial (having the same arity as the root of G).*

It is not difficult to see that, provided the arities match, we can replace a leaf of a sum-product circuit $\mathcal{C}$ with a sum-product *subcircuit* $\mathcal{C}_0$ computing the same function (over the appropriate subdomain) without affecting $\mathcal{C}$'s output.

Finally, in the discussions below, we also need a way to talk about oracles in the context of a sum-product (sub)circuit. The structure of the protocol is such that the calls to the oracle will be made at the leaves of $\mathcal{C}$, and so we associate the oracle with the circuit's input.

Definition 11.7. *Given a sub-product (sub)circuit $\mathcal{C}$ and a list of oracles $O = \{O_i\}_{i \in [\ell]}$ with $O_i \colon \mathbb{F}^k \to \mathbb{F}$ for each $i \in [\ell]$, we say that $\mathtt{x}^O$ is an* **oracle input** *for $\mathcal{C}$ if it labels the leaves of $\mathcal{C}$ with oracle circuits: arithmetic circuits that may include oracle gates. An oracle gate is a gate labeled with the name of an oracle O_i; it has k inputs and a single output defined as the evaluation of O_i on its inputs.*

11.3 Sum-product subcircuits for layered arithmetic circuits

We show that the evaluation problem for a given layered arithmetic circuit can be reduced to a sum-product subcircuit, when given oracles for low-degree extensions of the wiring predicates. The reduction essentially consists of casting the *barebones protocol* of [GKR15, Section 3] as (the evaluation of) a sum-product subcircuit on an oracle input. The barebones protocol is an Interactive Proof that enables a verifier to check a statement of the form "$C(\vec{x}) = y$", where C is a layered arithmetic circuit of size S and depth D, in time $n \cdot \mathrm{poly}(D, \log S)$ and space $O(\log S)$, provided that the verifier has oracle access to low-degree extensions of the wiring predicates $\{\mathrm{add}_i, \mathrm{mul}_i\}_{i \in \{1,\ldots,D\}}$ for the circuit C. The arithmetization of C underlying that protocol provides the basic intuition for how to 'program' a sum-product subcircuit to encode this computation.

Lemma 11.8 (sum-product subcircuits for layered arithmetic circuits)**.** *Let $\mathbb{F}$ be a finite field, $C \colon \mathbb{F}^n \to \mathbb{F}$ a layered arithmetic circuit of depth D and size S, $H \subseteq \mathbb{F}$, $m, m' \in \mathbb{N}$ and $F = (\hat{\mathrm{add}}_i, \hat{\mathrm{mul}}_i)_{i \in \{1,\ldots,D\}}$ any degree-δ extensions of the wiring predicates $\{\mathrm{add}_i, \mathrm{mul}_i\}_{i \in \{1,\ldots,D\}}$, with $\delta \leq \mathrm{polylog}(S)$. Then there exists a sum-product subcircuit $\mathcal{C}$, constructible in time $n \cdot \mathrm{poly}(D, \log S)$ and space $O(\log S)$, and oracle input $\mathtt{x}^F$ constructible in time $\mathrm{poly}(D, \log S)$ and space $O(\log S)$, such that $\mathcal{C}[\mathtt{x}^F](\vec{x}) = C(\vec{x})$, for all $\vec{x} \in \mathbb{F}^n$.*

Moreover, $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ where $|H| = \mathrm{poly}(D, \log S)$, $\delta_{\mathsf{in}} = \Theta(1)$, $\delta_{\mathsf{lf}} = \mathrm{poly}(D, \log S)$, $|V(G)| = \Theta(D)$, $\mathsf{arity}(G) = O(n + \log S / \log |\mathbb{F}|)$, $\mathsf{in}(G) = \Theta(1)$, $\mathsf{width}(G) = \Theta(1)$; and $\mathsf{space}(\mathtt{x}) = O(\log S)$.

Proof. Let $H \subseteq \mathbb{F}$ be such that $|\mathbb{F}|^{\Omega(1)} \leq |H| \leq \mathrm{poly}(D, \log S)$; $m \in \mathbb{N}$ be such that $S \leq |H|^m \leq \mathrm{poly}(S)$; $m' \in \mathbb{N}$ be such that $n \leq |H|^{m'} \leq n \cdot \mathrm{poly}(D, \log S)$.

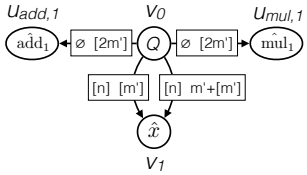


Figure 4: Sum-product circuit and its input for layered circuit of depth 1.

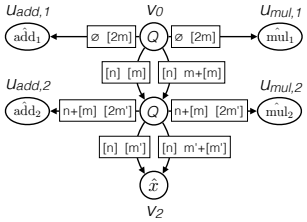


Figure 5: Sum-product circuit and its input for layered circuit of depth 2.

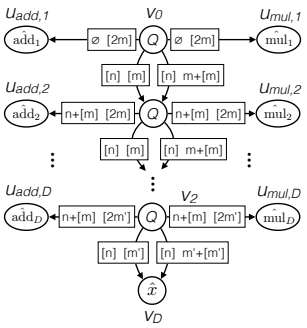


Figure 6: Sum-product circuit and its input for layered circuit of depth D.

First, we use D, m, m' to construct an ari-graph $G = (V, E, \rho, \sigma)$ as follows:

$$V := \{v_i\}_{i\in\{0,\dots,D\}} \cup \{u_{\text{add}_i}, u_{\text{mul}_i}\}_{i\in\{1,\dots,D\}}$$
$$E := \{e_i, e'_i = (v_i, v_{i+1})\}_{i\in\{0,\dots,D-1\}} \cup \{(v_i, u_{f_{i+1}}) : f \in \{\text{add}, \text{mul}\}\}_{i\in\{0,\dots,D-1\}}$$
$$\rho_e := \begin{cases} \varnothing & \text{if } e = (v_0, u_{f_1}) \text{ for } f \in \{\text{add}, \text{mul}\}, \text{ or} \\ n + [m] & \text{if } e = (v_i, u_{f_{i+1}}) \text{ for } f \in \{\text{add}, \text{mul}\} \text{ and } i \in \{1, \dots, D-1\}, \\ [n] & \text{otherwise;} \end{cases}$$
$$\sigma_e := \begin{cases} [2m] & \text{if } e = (v_i, u_{f_{i+1}}) \text{ for } f \in \{\text{add}, \text{mul}\} \text{ and } i \in \{0, \dots, D-2\}, \\ [m] & \text{if } e = e_i \text{ for some } i \in \{0, \dots, D-2\}, \\ m + [m] & \text{if } e = e'_i \text{ for some } i \in \{0, \dots, D-2\}, \\ [2m'] & \text{if } e = (v_{D-1}, u_{f_D}) \text{ for } f \in \{\text{add}, \text{mul}\}, \\ [m'] & \text{if } e = e_{D-1}, \text{ or} \\ m' + [m'] & \text{if } e = e'_{D-1}. \end{cases}$$

Note that E is a multiset and contains, for every $i \in \{0, \dots, D-1\}$, two distinct edges, e_i and e'_i, from v_i to v_{i+1} (with different projection labels). The root of G is v_0, and its leaves are v_D and $\{u_{\text{add}_i}, u_{\text{mul}_i}\}_{i\in\{1,\dots,D\}}$.

Next, we construct the sum-product circuit $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ where $\delta_{\mathsf{in}} := 3$, $\delta_{\mathsf{lf}} := \max(\delta, |H|)$, and C labels internal vertices of G as follows. For every $i \in \{0, \dots, D-1\}$, $C_{v_i} := X_1 \cdot (Y_1 + Y_2) + X_2 \cdot Y_1 \cdot Y_2$, where Y_1, Y_2 correspond to e_i, e'_i respectively, and X_1, X_2 correspond to $(v_i, u_{\text{add}_i}), (v_i, u_{\text{mul}_i})$ respectively.

Finally, we construct the input $\mathtt{x}$ for $\mathcal{C}$ as follows:

- for all $f \in \{\text{add}, \text{mul}\}$, $\mathtt{x}_{u_{f_1}}$ is the oracle circuit that outputs $\hat{f}_1(\vec{0}, \vec{\omega}_1, \vec{\omega}_2)$ on input $(\vec{\omega}_1, \vec{\omega}_2)$;
- for all $i \in \{2, \dots, D\}$ and $f \in \{\text{add}, \text{mul}\}$, $\mathtt{x}_{u_{f_i}}$ is the oracle circuit that outputs $\hat{f}_i(\vec{z}, \vec{\omega}_1, \vec{\omega}_2)$ on input $(\vec{z}, \vec{\omega}_1, \vec{\omega}_2)$;
- $\mathtt{x}_{v_D}$ is the polynomial $V_D(\vec{x}, \vec{z}) := \sum_{\vec{\beta}\in H^{m'}} I_{H^{m'}}(\vec{z}, \vec{\beta}) x_{\alpha'(\vec{\beta})}$ (with $x_i := 0$ for $i > n$), which has individual degree less than $|H|$, that is the low-degree extension of V_D and can be computed in time $|H|^{m'} \cdot \text{poly}(|H|, m') \le n \cdot \text{poly}(D, \log S)$ and space $O(m' \cdot \log |H|) \le \log n + \text{polylog}(D, \log S) \le O(\log S)$.

See Figure 4, Figure 5, Figure 6 for diagrams of this sum-product circuit and input for depth 1, 2, D respectively.

We are left to argue correctness of the reduction: it is easy to see that $v_D[\mathtt{x}^F](\vec{x}, \vec{z}) = V_D(\vec{x}, \vec{z})$ for every $\vec{x} \in \mathbb{F}^n$ and $\vec{z} \in H^{m'}$; hence, for every $i \in \{1, \dots, D-1\}$, $v_i[\mathtt{x}^F](\vec{x}, \vec{z}) = V_i(\vec{x}, \vec{z})$ for every $\vec{x} \in \mathbb{F}^n$ and $\vec{z} \in H^m$. We conclude that $\mathcal{C}[\mathtt{x}^F] = v_0[\mathtt{x}^F] = V_0$ (as functions in $\vec{x}$), as claimed. □

11.4 Sum-product subcircuits for small-space Turing machines

We show that small-space Turing machine computations can be reduced to the evaluation of sum-product subcircuits whose size is polynomially related to the space bound. This theorem is the analogue of [GKR15, Theorem 4.4] for our

framework: we use Lemma 11.8 and then instantiate the oracle input with explicit arithmetic circuits.

To prove Theorem 11.1 we need to construct a sum-product subcircuit whose value is a function $f\colon \mathbb{F}^n \to \mathbb{F}$ computable by a Turing machine in small space. However, if we simply apply Lemma 11.8 to a circuit given by [GKR15, Lemma 4.3], then we only obtain sum-product subcircuits for *boolean* functions computable in small space. We therefore apply [GKR15, Lemma 4.3] to the language $\mathscr{L}_f := \{(\vec{x}, a) : f(\vec{x}) = a\}$, which by Lemma 11.8 yields a subcircuit whose value is a function $f'\colon \mathbb{F}^n \times \mathbb{F} \to \{0,1\}$, the indicator function of $\mathscr{L}_f$. Then it holds that $\sum_{a\in\mathbb{F}} a \cdot f'(x,a) = f(x)$, and this summation can be implemented by adding an extra vertex to the subcircuit.

Lemma 11.9 (sum-product subcircuits for small-space Turing machines)**.** *Let $s\colon \mathbb{N} \to \mathbb{N}$ be a space function with $s(n) = \Omega(\log n)$, H an extension field of $\mathbb{F}_2$, and $\mathbb{F}$ an extension field of H with $|H| = \mathrm{poly}(s(n))$ and $|\mathbb{F}| = \mathrm{poly}(|H|)$. Let $f\colon \mathbb{F}^n \to \mathbb{F}$ be a function computable by a non-deterministic Turing machine M in space $s(n)$ in the following sense: M accepts $(\vec{x}, a) \in \mathbb{F}^n \times \mathbb{F}$ if and only if $f(\vec{x}) = a$. Then there exist a sum-product subcircuit $\mathcal{C}$ and input $\mathtt{x}$ for $\mathcal{C}$ such that $\mathcal{C}[\mathtt{x}](\vec{x}) = f(\vec{x})$ for all $x \in \mathbb{F}^n$; $\mathcal{C}$ is constructible in time $n \cdot \mathrm{poly}(s(n))$ and space $O(s(n))$, and $\mathtt{x}$ is constructible in time $n \cdot \mathrm{poly}(s(n))$ and space $O(s(n))$.*

Moreover, $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ where $\delta_{\mathsf{in}} = \Theta(1)$, $\delta_{\mathsf{lf}} = \mathrm{poly}(s(n))$, $|V(G)| = O(s^2(n))$, $\mathsf{arity}(G) = O(n + s(n)/\log|\mathbb{F}|)$, $\mathsf{in}(G) = \Theta(1)$, $\mathsf{width}(G) = \Theta(1)$; and $\mathsf{space}(\mathtt{x}) = O(s(n))$.

Proof. Fix an input length n, and let $k := \log|\mathbb{F}|/\log|H|$ (k is an integer because $\mathbb{F}$ is an extension field of H). Let $\gamma\colon H^k \to \mathbb{F}$ be the trivial isomorphism, and let $\mathscr{L} := \{(\vec{x}, \vec{z}) : f(\vec{x}) = \gamma(\vec{z})\}$. Note that $\mathscr{L}$ is computable by a non-deterministic Turing machine in space $O(s(n))$ because (i) f is and (ii) $\gamma(\vec{z})$ can be implemented at the bit level as the identity function. By [GKR15, Lemma 4.3], $\mathscr{L}$ can be computed by a layered arithmetic circuit C over $\mathbb{F}$ of size $S(n) = \mathrm{poly}(2^{s(n)})$ and depth $D(n) = O(s^2(n))$; moreover, one can generate arithmetic circuits for and evaluate some low-degree extensions $\hat{\mathrm{add}}_i, \hat{\mathrm{mul}}_i$ of the wiring predicates in time $\mathrm{poly}(s(n))$ and space $O(\log s(n))$. The degree of these extensions is $\mathrm{poly}(s(n)) = \mathrm{polylog}(S(n))$, independent of $\mathbb{F}$ (which is crucial for soundness).

Note that H is of size $\mathrm{poly}(s(n)) = \mathrm{poly}(D(n), \log S(n))$. By Lemma 11.8, we can construct a sum-product subcircuit $\mathcal{C}' = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ and oracle input $\mathtt{x}^F$ such that $\mathcal{C}'[\mathtt{x}^F](\vec{x}, \vec{z}) = C(\vec{x}, \vec{z})$ for every $\vec{x} \in \mathbb{F}^n$ and $\vec{z} \in H^k$. We replace the oracle gates in $\mathtt{x}^F$ with the explicit circuits for $\hat{\mathrm{add}}_i, \hat{\mathrm{mul}}_i$ to obtain a non-oracle input $\mathtt{x}'$.

Now observe that

$$f(\vec{x}) = \sum_{\vec{z}\in H^k} C(\vec{x}, \vec{z}) \cdot \gamma(\vec{z}) = \sum_{\vec{z}\in H^k} \mathcal{C}'[\mathtt{x}](\vec{x}, \vec{z}) \cdot \hat{\gamma}(\vec{z})$$

where $\hat{\gamma}\colon \mathbb{F}^k \to \mathbb{F}$ is the unique $\mathbb{F}$-linear function such that $\hat{\gamma}(\vec{z}) = \gamma(\vec{z})$ for $\vec{z} \in H^k$. Note that $\hat{\gamma}$ is computable by an arithmetic circuit of size $O(k)$, and can be evaluated in space $O(k + \log|\mathbb{F}|) = O(\log|\mathbb{F}|)$.

The sum-product subcircuit $\mathcal{C}$ is obtained from $\mathcal{C}'$ by adding a vertex v to G labeled with $X \cdot Y$ and a leaf vertex u_γ, and two edges $e := (v, r_G)$ and $e' := (v, u_\gamma)$ with labels $(\rho_e, \sigma_e) := ([n], [k])$ and $(\rho_{e'}, \sigma_{e'}) := (\varnothing, [k])$. The input $\mathtt{x}$ is obtained from $\mathtt{x}'$ by mapping u_γ to the arithmetic circuit computing $\hat{\gamma}$. □

11.5 Proof of Theorem 11.1

Let $\mathscr{L}$ be a language decidable by a family of $O(\log S(n))$-space uniform boolean circuits of size $S(n)$ and depth $D(n)$, and let $b(n)$ be a query bound function. The prover and verifier receive an input $\vec{x} \in \{0,1\}^n$. Let C be the boolean circuit (i.e., arithmetic circuit over $\mathbb{F}_2$) for this input size; we assume that C is layered with fan-in 2, as in Definition 11.3. (Any circuit can be efficiently converted into this form, with only a quadratic increase in size and a $\log S(n)$-factor increase in depth, which does not affect the theorem's statement.)

Let H be an extension field of $\mathbb{F}_2$ and $\mathbb{F}$ an extension field of H such that $|\mathbb{F}| = \mathrm{poly}(D(n), \log S(n))$ and $|H| = |\mathbb{F}|^{\Omega(1)}$. Define $m := \lceil \log S(n)/\log|H| \rceil$ and $m' := \lceil \log n/\log|H| \rceil$. Throughout we fix $\delta_{\mathsf{in}} = \Theta(1)$ and $\delta_{\mathsf{lf}} = \mathrm{poly}(D(n), \log S(n))$, as these settings suffice for all the (sub)circuits that we construct.

Part A: efficiently computable sum-product subcircuits for low-degree extensions of the wiring predicates. By [GKR15, Claim 4.6], for every $i \in \{1, \ldots, D(n)\}$, the low-degree extensions $\hat{\mathrm{add}}_i, \hat{\mathrm{mul}}_i\colon \mathbb{F}^m \to \mathbb{F}$ of C's wiring predicates can be computed by a Turing machine in space $O(\log S(n))$. The machine takes as input 1^n, H, $\mathbb{F}$, m, m', i, $f \in \{\mathrm{add}, \mathrm{mul}\}$, and $(\vec{a}, \vec{b}, \vec{c}) \in \mathbb{F}^{3m}$, and outputs $\hat{f}_i(\vec{a}, \vec{b}, \vec{c})$. Here we consider all inputs besides $(\vec{a}, \vec{b}, \vec{c})$ as hard-coded in the machine. By Lemma 11.9, there exist sum-product subcircuits $\mathcal{C}_{f_i} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G_{f_i}, C_{f_i})$ and inputs $\mathtt{x}_{f_i}$ (for every $f \in \mathrm{add}, \mathrm{mul}$ and $i \in \{1, \ldots, D\}$) such that $\mathcal{C}_{f_i}[\mathtt{x}_{f_i}](\vec{a}, \vec{b}, \vec{c}) = \hat{f}_i(\vec{a}, \vec{b}, \vec{c})$ for all $(\vec{a}, \vec{b}, \vec{c}) \in \mathbb{F}^{3m}$.

For every $f \in \text{add}, \text{mul}$ and $i \in \{1, \ldots, D\}$, we have $|V(G_{f_i})| = \text{poly}(\log S(n))$, $\mathsf{arity}(G_{f_i}) = O(\log S(n)/\log|\mathbb{F}|)$, $\mathsf{in}(G_{f_i}) = \Theta(1)$, $\mathsf{width}(G_{f_i}) = \Theta(1)$, and $\mathsf{space}(\mathtt{x}_{f_i}) = O(\log S(n))$.

Part B: sum-product circuit with oracle input for $C(\vec{x})$. We now invoke Lemma 11.8 on C to obtain a sum-product subcircuit $\mathcal{C}' = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G', C')$ and oracle input $\mathtt{x}^F$. For the theorem, we need to transform this into a sum-product *circuit* (i.e., where the root has arity 0). To do this, we will modify the projections so that the input $\vec{x}$ is no longer 'carried down', and then 'hard-code' it into $\mathtt{x}^F$.

Let ρ' be given by defining, for every $e \in E'$,

$$\rho'_e := \begin{cases} \{1, \ldots, m\} & \text{if } e = (v_i, u_{f_{i+1}}) \text{ for } f \in \{\text{add}, \text{mul}\} \text{ and } i \in \{1, \ldots, D-1\} \text{ or} \\ \varnothing & \text{otherwise.} \end{cases}$$

Observe that in $G'' := (V(G'), E(G'), \rho', \sigma(G'))$, the root v_0 has $\mathsf{arity}(v_0) = 0$, and so G'' is an ari-graph. Let the input $\mathtt{x}'^F$ be identical to $\mathtt{x}^F$ except that $\mathtt{x}'^F_{v_D}(\vec{Z}) := \mathtt{x}^F_{v_D}(\vec{x}, \vec{Z})$; that is, we 'hard-code' the input $\vec{x}$ to the circuit C into the input $\mathtt{x}'^F$ of the sum-product circuit (as in the original GKR protocol). This can only reduce the degree of $\mathtt{x}'^F_{v_D}$. Now let $\mathcal{C}'' := (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G'', C')$; it holds that that $\mathcal{C}''$ is a sum-product circuit and $\mathcal{C}''[\mathtt{x}'^F] = \mathcal{C}'[\mathtt{x}^F](\vec{x})$.

We obtain the following parameters: $|V(G'')| = \Theta(D(n))$, $\mathsf{arity}(G'') = O(\log S(n)/\log|\mathbb{F}|)$, $\mathsf{in}(G'') = \Theta(1)$, $\mathsf{width}(G'') = \Theta(1)$; and $\mathsf{space}(\mathtt{x}'^F) = O(\log S(n))$.

Part C: composing sum-product subcircuits. The final sum-product circuit $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ is constructed as follows. The ari-graph G is obtained from G'' by replacing each leaf node u_{f_i} with the ari-graph G_{f_i}, every $f \in \text{add}, \text{mul}$ and $i \in \{1, \ldots, D\}$. The vertex label C is the union of the vertex labels C' and $\{C_{f_i}\}_{f \in \{\text{add},\text{mul}\}, i \in \{1,\ldots,D\}}$.

The input $\mathtt{x}$ for $\mathcal{C}$ is equals the union of $\{\mathtt{x}_{f_i}\}_{f \in \{\text{add},\text{mul}\}, i \in \{1,\ldots,D\}}$ and the mapping $\mathtt{x}_{v_D} := \mathtt{x}'^F_{v_D}$ (for $v_D \in G''$).

Given the above definitions, one can verify that $\mathcal{C}[\mathtt{x}] = C(\vec{x})$.

Moreover, by inspection: $|V(G)| = D(n) \cdot \text{poly}(\log S(n))$, $\mathsf{arity}(G) = O(\log S(n)/\log|\mathbb{F}|)$, $\mathsf{in}(G) = \Theta(1)$, $\mathsf{width}(G) = \text{poly}(\log S(n))$, and $\mathsf{space}(\mathtt{x}) = O(\log S(n))$.

Part D: invoke PZK IPCP for sum-product circuit evaluation. All oracles have now been instantiated, we can now rely on our construction of perfect zero knowledge Interactive PCPs for sum-product circuit evaluation problems. More precisely, the prover and verifier construct the sum-product circuit $\mathcal{C}$ and its input $\mathtt{x}$ as above, in time $n \cdot \text{poly}(D(n), \log S(n))$ and space $O(\log S(n))$. (While this amount of space is not enough to store the whole circuit at once, the verifier can construct each part as needed.) The prover and verifier then engage in the protocol of Theorem 9.1 on the input $(\mathcal{C}, 1, \mathtt{x})$. Correctness comes from the fact that $(\mathcal{C}, 1, \mathtt{x}) \in \mathscr{L}_{\text{SPCE}}$ if and only if $C(\vec{x}) = 1$.

Plugging the parameters above into Theorem 9.1 yields the parameters claimed by Theorem 11.1, with the exception of the verifier space bound. A direct computation only shows that the verifier runs in space $\text{poly}(\log S(n))$, because $\mathsf{width}(G) = \text{poly}(\log S(n))$. Yet, since $\mathcal{C}$ is 'treelike' in the sense that the subcircuits can be handled independently, the evaluation protocol need only consider a constant number of vertices at any one time, at the expense of a $\text{poly}(\log S(n))$-factor increase in the round complexity. Doing so yields the claimed $O(\log S(n))$ space complexity of the verifier.

Acknowledgments

The authors would like to thank Eli Ben-Sasson for numerous enlightening conversations in early stages of this work. The authors also thank Thomas Vidick for suggesting that we use our techniques to obtain a zero knowledge analogue of [BFL91; BFLS91], as captured by Theorem 7.2. The authors thank Tom Gur for comments on the writeup.

A Algebraic query complexity of polynomial summation: details

In this section we prove the results whose statements appear in Section 5.

A.1 Proof of Theorem 5.1

First, since Z has individual degree at most d in $\vec{X}$, we can rewrite any such linear combination in the following way:

$$\sum_{\vec{\alpha}\in L^m} C_{\vec{\alpha},i} \sum_{\vec{y}\in G^k} Z(\vec{\alpha},\vec{y}) = \sum_{\vec{\alpha}\in L^m} C_{\vec{\alpha},i} \sum_{\vec{\beta}\in K^m} b_{\vec{\beta},\vec{\alpha}} \sum_{\vec{y}\in G^k} Z(\vec{\alpha},\vec{y}) = \sum_{\vec{\alpha}\in K^m} C'_{\vec{\alpha},i} \sum_{\vec{y}\in G^k} Z(\vec{\alpha},\vec{y}) = \sum_{\vec{q}\in S} D_{\vec{q},i} Z(\vec{q}) \ ,$$

where $C' := BC$. If $d' = |G| - 2$ then the bound is trivial. Otherwise, let H be some arbitrary subset of G of size $\min\{d' - |G| + 2, |G|\}$. Let $P_0 \subseteq \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq |H|-1}_{1,\ldots,k}]$ be such that for all $p \in P_0$ and for all $\vec{q} \in S$, $p(\vec{q}) = 0$. Since this is at most S linear constraints, P_0 has dimension at least $(d+1)^m|H|^k - |S|$. Let $B_0 \in \mathbb{F}^{n\times(d+1)^m|H|^k}$ be a matrix whose rows form a basis for the vector space $\{\big(p(\vec{\alpha},\vec{\beta})\big)_{\vec{\alpha}\in K^m,\vec{\beta}\in H^k} : p \in P_0\}$ of evaluations of polynomials in P_0 on $K^m \times H^k$; we have $n \geq (d+1)^m|H|^k - |S|$. By an averaging argument there exists $\vec{\beta}_0 \in H^k$ such that the submatrix $B_{\vec{\beta}_0}$ consisting of columns $(\vec{\alpha},\vec{\beta}_0)$ of B_0 for each $\vec{\alpha} \in K^m$ has rank at least $(d+1)^m - |S|/|H|^k$.

Let $q \in \mathbb{F}[Y^{\leq |G|-1}_{1,\ldots,k}]$ be the polynomial such that $q(\vec{\beta}_0) = 1$, and $q(\vec{y}) = 0$ for all $\vec{y} \in G^k - \{\vec{\beta}_0\}$. For arbitrary $p \in P_0$, let $Z(\vec{X},\vec{Y}) := q(\vec{Y})p(\vec{X},\vec{Y}) \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq |H|+|G|-2}_{1,\ldots,k}]$. Observe that our choice of H ensures that the degree of Z in $\vec{Y}$ is at most d'. Then for all $i \in \{1,\ldots,\ell\}$, it holds that

$$\sum_{\vec{\alpha}\in K^m} C'_{\vec{\alpha},i} \sum_{\vec{y}\in G^k} Z(\vec{\alpha},\vec{y}) = \sum_{\vec{\alpha}\in K^m} C'_{\vec{\alpha},i} \cdot p(\vec{\alpha},\vec{\beta}_0) = \sum_{\vec{q}\in S} D_{\vec{q},i} \cdot Z(\vec{\alpha},\vec{y}) = 0 \ .$$

Thus the column space of C' is contained in the null space of $B_{\vec{\beta}_0}$, and so the null space of $B_{\vec{\beta}_0}$ has rank at least $\mathrm{rank}(C')$. Hence $(d+1)^m - \mathrm{rank}(C') \geq \mathrm{rank}(B_{\vec{\beta}_0}) \geq (d+1)^m - |S|/|H|^k$, so $|S| \geq \mathrm{rank}(C') \cdot |H|^k$, which yields the theorem. □

A.2 Proof of Corollary 5.3

We will need a simple fact from linear algebra: that 'linear independence equals statistical independence'. That is, if we sample an element from a vector space and examine some subsets of its entries, these distributions are independent if and only if there does not exist a linear dependence between the induced subspaces. The formal statement of the claim is as follows; its proof is deferred to the end of this subsection.

Claim A.1. *Let $\mathbb{F}$ be a finite field and D a finite set. Let $V \subseteq \mathbb{F}^D$ be an $\mathbb{F}$-vector space, and let $\vec{v}$ be a random variable which is uniform over V. For any subdomains $S, S' \subseteq D$, the restrictions $\vec{v}|_S$ and $\vec{v}|_{S'}$ are statistically dependent if and only if there exist constants $(c_i)_{i\in S}$ and $(d_i)_{i\in S'}$ such that:*

- *there exists $\vec{w} \in V$ such that $\sum_{i\in S} c_i w_i \neq 0$, and*
- *for all $\vec{w} \in V$, $\sum_{i\in S} c_i w_i = \sum_{i\in S'} d_i w_i$.*

Now observe that

$$\left\{\Big(\big(Z(\vec{\gamma})\big)_{\vec{\gamma}\in\mathbb{F}^{m+k}}, \big(\sum_{\vec{y}\in G^k} Z(\vec{\alpha},\vec{y})\big)_{\vec{\alpha}\in\mathbb{F}^m}\Big) : Z \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}, Y^{\leq d'}_{1,\ldots,k}]\right\}$$

is an $\mathbb{F}$-vector space with domain $\mathbb{F}^{m+k} \cup \mathbb{F}^m$. Consider subdomains $\mathbb{F}^m$ and S. Since $|S| < |G|^k$, by Theorem 5.1 there exist no constants $(c_{\vec{\alpha}})_{\alpha\in\mathbb{F}^m}$, $(d_{\vec{\gamma}})_{\vec{\gamma}\in S}$ such that the conditions of the claim hold. This completes the proof.

Proof of Claim A.1. For arbitrary $\vec{x} \in \mathbb{F}^S, \vec{x}' \in \mathbb{F}^{S'}$, we define the quantity

$$p_{\vec{x},\vec{x}'} := \Pr_{\vec{v} \in V}[\vec{v}|_S = \vec{x} \wedge \vec{v}|_{S'} = \vec{x}'] \ .$$

Let $d := \dim(V)$, and let $B \in \mathbb{F}^{D \times d}$ be a basis for V. Let $B_S \in \mathbb{F}^{S \times d}$ be B restricted to rows corresponding to elements of S, and let $B_{S'}$ be defined likewise. Finally, let $B_{S,S'} \in \mathbb{F}^{(|S|+|S'|) \times d}$ be the matrix whose rows are the rows of B_S, followed by the rows of $B_{S'}$. Then

$$p_{\vec{x},\vec{x}'} = \Pr_{\vec{z} \in \mathbb{F}^d}[B_{S,S'} \cdot \vec{z} = (\vec{x}, \vec{x}')] \ .$$

One can verify that, for any matrix $A \in \mathbb{F}^{m \times n}$,

$$\Pr_{\vec{z} \in \mathbb{F}^n}[A\vec{z} = \vec{b}] = \begin{cases} \mathbb{F}^{-\operatorname{rank}(A)} & \text{if } \vec{b} \in \operatorname{colsp}(A), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The column space $\operatorname{colsp}(B_{S,S'}) \subseteq \operatorname{colsp}(B_S) \times \operatorname{colsp}(B_{S'})$, and equality holds if and only if $\operatorname{rank}(B_{S,S'}) = \operatorname{rank}(B_S) + \operatorname{rank}(B_{S'})$. It follows that $p_{\vec{x},\vec{x}'} = \Pr_{\vec{v} \in V}[\vec{v}|_S = \vec{x}] \cdot \Pr_{\vec{v} \in V}[\vec{v}|_{S'} = \vec{x}']$ if and only if $\operatorname{rank}(B_{S,S'}) = \operatorname{rank}(B_S) + \operatorname{rank}(B_{S'})$. By the rank-nullity theorem and the construction of $B_{S,S'}$, this latter condition holds if and only if $\operatorname{nul}(B_{S,S'}^T) \subseteq \operatorname{nul}(B_S^T) \times \operatorname{nul}(B_{S'}^T)$. To conclude the proof, it remains only to observe that the condition in the claim is equivalent to the existence of vectors $\vec{c} \in \mathbb{F}^S$, $\vec{d} \in \mathbb{F}^{S'}$ such that $\vec{c} \notin \operatorname{nul}(B_S^T)$ but $(\vec{c}, -\vec{d}) \in \operatorname{nul}(B_{S,S'}^T)$. □

A.3 Upper bounds

In this section we show that in certain cases the degree constraints in Theorem 5.1 are tight.

A.3.1 The case of multilinear polynomials

The first result is for the case of multivariate polynomials over any finite field, where $H \subseteq \mathbb{F}$ is arbitrary. The proof is a simple extension of a proof due to [JKRS09] for the case $H = \{0, 1\}$.

Theorem A.2 (Multilinear Polynomials)**.** *Let $\mathbb{F}$ be a finite field, H a subset of $\mathbb{F}$, and $\gamma := \sum_{\alpha \in H} \alpha$. For every $P \in \mathbb{F}[X_{1,\ldots,m}^{\leq 1}]$ (i.e., for every m-variate multilinear polynomial P) it holds that*

$$\sum_{\vec{\alpha} \in H^m} P(\vec{\alpha}) = \begin{cases} P\left(\frac{\gamma}{|H|}, \ldots, \frac{\gamma}{|H|}\right) \cdot |H|^m & \text{if } \operatorname{char}(\mathbb{F}) \nmid |H| \\ \kappa \cdot \gamma^m & \text{if } \operatorname{char}(\mathbb{F}) \mid |H| \end{cases},$$

where κ is the coefficient of $X_1 \cdots X_m$ in P.

Proof. First suppose that $\operatorname{char}(\mathbb{F})$ does not divide $|H|$. Let $\vec{\alpha}$ be uniformly random in H^m; in particular, α_i and α_j are independent for $i \neq j$. For every monomial $m(\vec{X}) = X_1^{e_1} \cdots X_m^{e_m}$ with $e_1, \ldots, e_m \in \{0, 1\}$,

$$\mathbb{E}[M(\vec{\alpha})] = \mathbb{E}[\alpha_1^{e_1} \cdots \alpha_m^{e_m}] = \mathbb{E}[\alpha_1^{e_1}] \cdots \mathbb{E}[\alpha_m^{e_m}] = \mathbb{E}[\alpha_1]^{e_1} \cdots \mathbb{E}[\alpha_m]^{e_m} = M(\mathbb{E}[\alpha_1], \ldots, \mathbb{E}[\alpha_m]) \ .$$

Since P is a linear combination of monomials, $\mathbb{E}[P(\vec{\alpha})] = P(\mathbb{E}[\vec{\alpha}])$. Each α_i is uniformly random in H, so $\mathbb{E}[\alpha_i] = \frac{1}{|H|}\sum_{\alpha \in H} \alpha = \frac{\gamma}{|H|}$, and thus $P(\mathbb{E}[\vec{\alpha}]) = P(\frac{\gamma}{|H|}, \ldots, \frac{\gamma}{|H|})$, which implies that $\mathbb{E}[P(\vec{\alpha})] = P(\frac{\gamma}{|H|}, \ldots, \frac{\gamma}{|H|})$. To deduce the claimed relation, it suffices to note that $\mathbb{E}[P(\vec{\alpha})] = \frac{1}{|H|^m}\sum_{\vec{\alpha} \in H^m} P(\vec{\alpha})$.

Next suppose that $\operatorname{char}(\mathbb{F})$ divides $|H|$. For every monomial $m(\vec{X}) = X_1^{e_1} \cdots X_m^{e_m}$ with $e_1, \ldots, e_m \in \{0, 1\}$:

- if there exists $j \in [m]$ such that $e_j = 0$ then

$$\sum_{\vec{\alpha} \in H^m} M(\vec{\alpha}) = |H| \sum_{\alpha_1, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_m \in H} \alpha_1^{e_1} \cdots \alpha_{j-1}^{e_{j-1}} \alpha_{j+1}^{e_{j+1}} \cdots \alpha_m^{e_m} = 0 \ .$$

- if instead $e_1 = \cdots = e_m = 1$ then

$$\sum_{\vec{\alpha} \in H^m} M(\vec{\alpha}) = \sum_{\vec{\alpha} \in H^m} \prod_{i=1}^{m} \alpha_i = \prod_{i=1}^{m} \sum_{\alpha_i \in H} \alpha_i = \left(\sum_{\alpha \in H} \alpha \right)^m \quad . \qquad \square$$

The following corollary shows that for prime fields of odd size, the value of $\sum_{\vec{\alpha} \in H^m} P(\vec{\alpha})$ can be computed efficiently for any $H \subseteq \mathbb{F}$ using at most a single query to P.

Corollary A.3. *Let $\mathbb{F}$ be a prime field of odd size, H a subset of $\mathbb{F}$, and $\gamma := \sum_{\alpha \in H} \alpha$. For every $P \in \mathbb{F}[X^{\leq 1}_{1,\ldots,m}]$ (i.e., for every m-variate multilinear polynomial P) it holds that*

$$\sum_{\vec{\alpha} \in H^m} P(\vec{\alpha}) = \begin{cases} P\left(\frac{\gamma}{|H|}, \ldots, \frac{\gamma}{|H|}\right) \cdot |H|^m & \text{if } \mathrm{char}(\mathbb{F}) \nmid |H| \\ 0 & \text{if } \mathrm{char}(\mathbb{F}) \mid |H| \end{cases} .$$

Proof. Theorem A.2 directly implies both cases. If $\mathrm{char}(\mathbb{F})$ does not divide $|H|$, then the claimed value is as in the theorem. If instead $\mathrm{char}(\mathbb{F})$ divides $|H|$, then it must be the case that $H = \mathbb{F}$, since $p := \mathrm{char}(\mathbb{F})$ equals $|\mathbb{F}|$; in this case, $\gamma = \sum_{\alpha \in H} \alpha = (p-1)p/2$, which is divisible by p since 2 must divide $p-1$ (as p is odd). $\square$

A.3.2 The case of subsets with group structure

In this section we show that if H is assumed to have some group structure, then few queries may suffice even for polynomials of degree greater than one. In particular, the following result shows that a single query suffices for $d \leq |H|$ when H is a multiplicative subgroup of $\mathbb{F}$.

Lemma A.4 (Multiplicative Groups)**.** *Let $\mathbb{F}$ be a field, H a finite multiplicative subgroup of $\mathbb{F}$, and m, d positive integers with $d < |H|$. For every $P \in \mathbb{F}[X^{\leq d}_{1,\ldots,m}]$,*

$$\sum_{\vec{\alpha} \in H^m} P(\vec{\alpha}) = P(0, \ldots, 0) \cdot |H|^m \quad .$$

Remark A.5. The hypothesis that $d < |H|$ is necessary for the lemma, as we now explain. Choose $H = \mathbb{K}^\times$, where $\mathbb{K}$ is a proper subfield of $\mathbb{F}$, $m = 1$, and $d = |H|$. Consider the polynomial $X^{|H|}$, which has degree at least d: $X^{|H|}$ vanishes on 0; however, $X^{|H|}$ evaluates to 1 everywhere on H so that its sum over H equals $|H| \neq 0$. (Note that if H is a multiplicative subgroup of $\mathbb{F}$ then $\mathrm{char}(\mathbb{F}) \nmid |H|$ because $|H|$ equals $\mathrm{char}(\mathbb{F})^k - 1$ for some positive integer k.)

Proof. The proof is by induction on the number of variables m. The base case is when $m = 1$, which we argue as follows. The group H is cyclic, because it is a (finite) multiplicative subgroup of a field; so let ω generate H. Writing $P(X_1) = \sum_{j=0}^{d} \beta_j X_1^j$ for some $\beta_0, \ldots, \beta_d \in \mathbb{F}$, we have

$$\sum_{\alpha_1 \in H} P(\alpha_1) = \sum_{i=0}^{|H|-1} P(\omega^i) = \sum_{i=0}^{|H|-1} \sum_{j=0}^{d} \beta_j \omega^{ij} = \sum_{j=0}^{d} \beta_j \sum_{i=0}^{|H|-1} (\omega^j)^i = \beta_0 |H| = f(0)|H| \quad ,$$

which proves the base case. The second-to-last equality follows from the fact that for every $\gamma \in H$,

$$\sum_{i=0}^{|H|-1} \gamma^i = \begin{cases} |H| & \text{if } \gamma = 1 \\ \frac{\gamma^{|H|}-1}{\gamma - 1} = 0 & \text{if } \gamma \neq 1 \end{cases} .$$

For the inductive step, assume the statement for any number of variables less than m; we now prove that it holds for m variables as well. Let P_α denote P with the variable X_1 fixed to α. Next, apply the inductive assumption below in

the second equality (with $m-1$ variables) and last one (with 1 variable), to obtain

$$\begin{aligned}\sum_{\vec{\alpha}\in H^m} P(\alpha_1,\ldots,\alpha_m) &= \sum_{\alpha_1\in H}\sum_{(\alpha_2,\ldots,\alpha_m)\in H^{m-1}} P_{\alpha_1}(\alpha_2,\ldots,\alpha_m)\\ &= |H|^{m-1}\sum_{\alpha_1\in H} P_{\alpha_1}(0^{m-1})\\ &= |H|^{m-1}\sum_{\alpha_1\in H} P(\alpha_1,0,\ldots,0)\\ &= |H|^m P(0,\ldots,0)\ ,\end{aligned}$$

as claimed. □

Lemma A.6 (Additive Groups)**.** *Let $\mathbb{F}$ be a field, H a finite additive subgroup of $\mathbb{F}$, and m, d positive integers with $d < |H|$. For every $P \in \mathbb{F}[X_{1,\ldots,m}^{\leq d}]$,*

$$\sum_{\vec{\alpha}\in H^m} P(\vec{\alpha}) = \kappa\cdot a_0^m\ ,$$

where κ is the coefficient of $X_1^{|H|-1}\cdots X_m^{|H|-1}$ in P, and a_0 is the (formal) linear term of the subspace polynomial $\prod_{h\in H}(X-h)$. In particular, if P has total degree strictly less than $m(|H|-1)$, then the above sum evaluates to 0.

Proof. Without loss of generality, let $d := |H|-1$. The proof is by induction on the number of variables m. When $m=1$, we have that $P(X)=\sum_{j=0}^d \beta_j X^j$ for some $\beta_0,\ldots,\beta_d\in\mathbb{F}$. Then

$$\sum_{\alpha\in H} P(\alpha) = \sum_{\alpha\in H}\sum_{j=0}^{d}\beta_j\alpha^j = \sum_{j=0}^{d}\beta_j\sum_{\alpha\in H}\alpha^j = \beta_d a_0$$

where the final equality follows by [BC99, Theorem 1], and the fact that $d=|H|-1$.

For the inductive step, assume the statement for $m-1$ variables; we now prove that it holds for m variables as well. Let P_α denote P with the variable X_1 fixed to α; we have $P_\alpha(X_2,\ldots,X_m)=\sum_{\vec{e}\in\{0,\ldots,d\}^m}\beta_{\vec{e}}\cdot\alpha^{e_1}X_2^{e_2}\ldots X_m^{e_m}$. Next, apply the inductive hypothesis below in the second equality (with $m-1$ variables) to obtain

$$\sum_{\vec{\alpha}\in H^m} P(\alpha_1,\ldots,\alpha_m) = \sum_{\alpha_1\in H}\sum_{(\alpha_2,\ldots,\alpha_m)\in H^{m-1}} P_{\alpha_1}(\alpha_2,\ldots,\alpha_m) = \sum_{\alpha_1\in H} a_0^{m-1}\kappa(\alpha_1)\ ,$$

where $\kappa(X_1) := \sum_{j=0}^d \beta_{(j,d,\ldots,d)}X_1^j$. Applying the hypothesis again for 1 variable yields

$$\sum_{\alpha_1\in H} a_0^{m-1}\kappa(\alpha_1) = a_0^m\cdot\beta_{(d,\ldots,d)}\ ,$$

and the claim follows. □

B Proof of Theorem 7.2 via sum-product circuits

In this section we re-prove Theorem 7.2 using sum-product circuits. In particular we reduce $\mathscr{R}_{\mathrm{O3SAT}}$ to $\mathscr{R}_{\mathrm{SPCS}}$ by constructing a sum-product circuit whose satisfaction encodes oracle 3-satisfiability. The reduction then yields Theorem 7.2 by the perfect zero knowledge IPCP for $\mathscr{R}_{\mathrm{SPCS}}$ given in Theorem 9.2.

Lemma B.1 ($\mathscr{R}_{\mathrm{O3SAT}} \to \mathscr{R}_{\mathrm{SPCS}}$)**.** *Let H be an extension field of $\mathbb{F}_2$ with $|H| \geq 8$, and $\mathbb{F}$ an extension field of H. There exist polynomial-time functions f, g such that for every $r, s \in \mathbb{N}$ and boolean formula $B\colon \{0,1\}^{r+3s+3} \to \{0,1\}$:*

1. *if $A\colon \{0,1\}^s \to \{0,1\}$ is such that $((r,s,B),A) \in \mathscr{R}_{\mathrm{O3SAT}}$ then*

$$\Pr_{\vec{x},\vec{y} \leftarrow \mathbb{F}^{r+3s}} \left[\Big((f(r,s,B,\vec{x},\vec{y}),0,\perp), g(A) \Big) \in \mathscr{R}_{\mathrm{SPCS}} \right] = 1 \ ;$$

2. *if $(r,s,B) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ then*

$$\Pr_{\vec{x},\vec{y} \leftarrow \mathbb{F}^{r+3s}} \left[\Big(f(r,s,B,\vec{x},\vec{y}),0,\perp \Big) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}}) \right] \leq \frac{r+3s}{|\mathbb{F}|} \ .$$

Moreover, $f(r,s,B,\vec{x},\vec{y})$ is a sum-product circuit $(\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ with $\delta_{\mathsf{in}} = O(|B| \cdot |H|)$, $\delta_{\mathsf{lf}} = O(|H|)$, $|V(G)| = \Theta(1)$, $\mathsf{arity}(G) = O((r+s)/\log|H|)$, $\mathsf{in}(G) = \Theta(1)$, $\mathsf{width}(G) = \Theta(1)$.

Proof. Recall that in Section 7 we reduced checking $((B,r,s),A) \in \mathscr{R}_{\mathrm{O3SAT}}$ to checking whether the following expression is the zero polynomial, for some polynomials g_1, g_2 which depend on the low-degree extension of A.

$$F(\vec{X},\vec{Y}) = \sum_{\substack{\vec{\alpha} \in H^{m_1} \\ \vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3 \in H^{m_2}}} g_1(\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3)) \prod_{i=1}^{r+3s} (1 + (X_i - 1)\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3)_i)$$
$$+ g_2(\hat{\gamma}_2(\vec{\beta}_1)) \prod_{i=1}^{r+3s} (1 + (Y_i - 1)\hat{\gamma}(\vec{\alpha},\vec{\beta}_1,\vec{\beta}_2,\vec{\beta}_3)_i) \ .$$

Fix $\vec{x},\vec{y} \in \mathbb{F}^{r+3s}$. We construct a sum-product circuit $\mathcal{C} = (\mathbb{F}, H, \delta_{\mathsf{in}}, \delta_{\mathsf{lf}}, G, C)$ whose value with explicit input $\perp$ (the empty map) and auxiliary input $\hat{A}$ is $F(\vec{x},\vec{y})$. Fix $\delta_{\mathsf{in}} := O(|B| \cdot |H|)$ and $\delta_{\mathsf{lf}} := |H|$. Let $G = (V,E,\sigma,\rho)$ be the ari-graph defined as follows:

$$\begin{aligned} V &:= \{v,u\} \ , \\ E &:= \{e_1,e_2,e_3 = (v,u)\} \ , \\ \rho_e &:= \varnothing \quad \text{for all } e \in E \ , \\ \sigma_{e_i} &:= m_1 + (i-1)m_2 + [m_2] \quad \text{for } i \in \{1,2,3\} \ . \end{aligned}$$

The vertex label C is given by

$$C_v := \left(\hat{B}(\vec{Y},Z_1,Z_2,Z_3) \cdot \prod_{i=1}^{r+3s+3} (1 + (x_i - 1))\hat{\gamma}(\vec{Y})_i \right) + \left(Z_1(1-Z_1) \cdot \prod_{i=1}^{r+3s} (1 + (y_i - 1))\hat{\gamma}(\vec{Y})_i \right) \ ,$$

where the variables Z_1, Z_2, Z_3 correspond to the edges e_1, e_2, e_3 respectively.

From the construction of the circuit, if $\mathtt{z}_u = \hat{A}(\hat{\gamma}_2(\cdot))$ then $\mathcal{C}[\mathtt{x},\mathtt{z}] = F(\vec{x},\vec{y})$. The stated parameters follow easily.

If $((r,s,B),A) \in \mathscr{R}_{\mathrm{O3SAT}}$ then $F(\vec{X},\vec{Y})$ is the zero polynomial. Hence $F(\vec{x},\vec{y}) = 0$ for all $\vec{x},\vec{y} \in \mathbb{F}^{r+3s}$. We conclude that if $((r,s,B),A) \in \mathscr{R}_{\mathrm{O3SAT}}$ then $((\mathcal{C},0,\mathtt{x}),\mathtt{z}) \in \mathscr{R}_{\mathrm{SPCS}}$ with probability 1 over the choice of $\vec{x},\vec{y}$.

If $(r,s,B) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ then there is no choice of $\hat{A}$ such that $F(\vec{X},\vec{Y})$ is the zero polynomial. Thus, for any choice of $\hat{A}$, $F(\vec{x},\vec{y}) = 0$ with probability at most $(r+3s)/|\mathbb{F}|$ over the choice of $\vec{x},\vec{y}$. We conclude that if $(r,s,B) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ then $(\mathcal{C},0,\mathtt{x}) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$ with probability at most $(r+3s)/|\mathbb{F}|$. □

Proof of Theorem 7.2. The protocol proceeds as follows. The prover and verifier determinstically (non-interactively) choose an extension field H of $\mathbb{F}_2$ and an extension field $\mathbb{F}$ of H such that $|\mathbb{F}| = \mathrm{poly}(|H|)$ and $|H| = \mathrm{poly}(|B|, \log b)$. The verifier chooses $\vec{x}, \vec{y} \in \mathbb{F}^{r+3s}$ uniformly at random, and sends them to the prover; the prover and verifier construct the sum-product circuit $\mathcal{C} := f(r, s, B, \vec{x}, \vec{y})$, and then engage in the protocol of Theorem 9.2 on the input $(\mathcal{C}, 0, \perp)$, with auxiliary input $g(A)$.

If $((B, r, s), A) \in \mathscr{R}_{\mathrm{O3SAT}}$ then $((\mathcal{C}, 0, \perp), g(A)) \in \mathscr{R}_{\mathrm{SPCS}}$ with probability 1, and so the verifier will accept with probability 1. If $(B, r, s) \notin \mathscr{L}(\mathscr{R}_{\mathrm{O3SAT}})$ then $(\mathcal{C}, 0, \perp) \in \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$ with probability at most $(r + 3s)/|\mathbb{F}|$. Also, if $(\mathcal{C}, 0, \perp) \notin \mathscr{L}(\mathscr{R}_{\mathrm{SPCS}})$, then the verifier accepts with probability at most $O(\delta_{\mathsf{in}}\delta_{\mathsf{lf}} \cdot \mathsf{in}(G) \cdot (\mathsf{arity}(G) + \alpha) \cdot |V(G)|/|\mathbb{F}|) = O(\frac{|H|^2 \cdot |B| \cdot (r+s+\log b)}{|\mathbb{F}| \cdot \log |H|})$. By our choices of the cardinality of H and $\mathbb{F}$, both of these probabilities are $O(1/|B|)$ so that, by a union bound, the probability that the verifier accepts is also $O(1/|B|)$, which is less than $1/2$.

The protocol as stated is not an IPCP because there is a round of interaction before the oracle is sent. However, observe that the oracle does not depend on the choice of $\vec{x}, \vec{y}$, and therefore can be sent before this interaction. □

References

[AH91] William Aiello and Johan Håstad. "Statistical Zero-Knowledge Languages can be Recognized in Two Rounds". In: *Journal of Computer and System Sciences* 42.3 (1991). Preliminary version appeared in FOCS '87., pp. 327–345.

[ALMSS98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (1998). Preliminary version in FOCS '92., pp. 501–555.

[AS03] Sanjeev Arora and Madhu Sudan. "Improved Low-Degree Testing and its Applications". In: *Combinatorica* 23.3 (2003). Preliminary version appeared in STOC '97., pp. 365–426.

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998). Preliminary version in FOCS '92., pp. 70–122.

[AW09] Scott Aaronson and Avi Wigderson. "Algebrization: A New Barrier in Complexity Theory". In: *ACM Transactions on Computation Theory* 1.1 (2009), 2:1–2:54.

[BC12] Nir Bitansky and Alessandro Chiesa. "Succinct Arguments from Multi-Prover Interactive Proofs and their Efficiency Benefits". In: *Proceedings of the 32nd Annual International Cryptology Conference*. CRYPTO '12. 2012, pp. 255–272.

[BC99] Nigel P. Byott and Robin J. Chapman. "Power Sums over Finite Subspaces of a Field". In: *Finite Fields and Their Applications* 5.3 (July 1999), pp. 254–265.

[BCFGRS16] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. *On Probabilistic Checking in Perfect Zero Knowledge*. Cryptology ePrint Archive, Report 2016/988. 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. 2016, pp. 31–60.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. "Non-Deterministic Exponential Time has Two-Prover Interactive Protocols". In: *Computational Complexity* 1 (1991). Preliminary version appeared in FOCS '90., pp. 3–40.

[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. 1991, pp. 21–32.

[BGGHKMR88] 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. 1988, pp. 37–56.

[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. 1988, pp. 113–131.

[BM88] László Babai and Shlomo Moran. "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes". In: *Journal of Computer and System Sciences* 36.2 (1988), pp. 254–276.

[BRV17] Itay Berman, Ron D. Rothblum, and Vinod Vaikuntanathan. *Zero-Knowledge Proofs of Proximity*. Cryptology ePrint Archive, Report 2017/114. 2017.

[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. 2004, pp. 298–309.

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

[CD98] Ronald Cramer and Ivan Damgård. "Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge be for Free?" In: *Proceedings of the 18th Annual International Cryptology Conference*. CRYPTO '98. 1998, pp. 424–441.

[CKLR11] Kai-Min Chung, Yael Kalai, Feng-Hao Liu, and Ran Raz. "Memory Delegation". In: *Proceeding of the 31st Annual Cryptology Conference*. CRYPTO '10. 2011, pp. 151–168.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical Verified Computation with Streaming Interactive Proofs". In: *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*. ITCS '12. 2012, pp. 90–112.

[CRR13] Ran Canetti, Ben Riva, and Guy N. Rothblum. "Refereed delegation of computation". In: *Information and Computation* 226 (2013), pp. 16–36.

[DFKNS92] 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. 1992, pp. 215–227.

[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. 1998, pp. 442–457.

[FGLSS91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating clique is almost NP-complete (preliminary version)". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. SFCS '91. 1991, pp. 2–12.

[FGLSS96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. "Interactive proofs and the hardness of approximating cliques". In: *Journal of the ACM* 43.2 (1996). Preliminary version in FOCS '91., pp. 268–292.

[FRS88] Lance Fortnow, John Rompel, and Michael Sipser. "On the Power of Multi-Prover Interactive Protocols". In: *Theoretical Computer Science*. 1988, pp. 156–161.

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[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. 1989, pp. 526–544.

[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. 1987, pp. 204–209.

[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. 2010, pp. 173–190.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The knowledge complexity of interactive proof systems". In: *SIAM Journal on Computing* 18.1 (1989). Preliminary version appeared in STOC '85., pp. 186–208.

[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. 2015, pp. 133–142.

[GS06] Oded Goldreich and Madhu Sudan. "Locally testable codes and PCPs of almost-linear length". In: *Journal of the ACM* 53 (4 2006). Preliminary version in STOC '02., pp. 558–655.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2001. ISBN: 978-0-521-03536-1.

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

[IY87] Russell Impagliazzo and Moti Yung. "Direct Minimum-Knowledge Computations". In: *Proceedings of the 7th Annual International Cryptology Conference*. CRYPTO '87. 1987, pp. 40–51.

[JKRS09] Ali Juma, Valentine Kabanets, Charles Rackoff, and Amir Shpilka. "The Black-Box Query Complexity of Polynomial Summation". In: *Computational Complexity* 18.1 (2009), pp. 59–79.

[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. 1997, pp. 496–505.

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

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. "How to delegate computations: the power of no-signaling proofs". In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. STOC '14. 2014, pp. 485–494.

[Kil92] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC '92. 1992, pp. 723–732.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39.4 (1992), pp. 859–868.

[Mei13] Or Meir. "IP = PSPACE Using Error-Correcting Codes". In: *SIAM Journal on Computing* 42.1 (2013), pp. 380–403.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.

[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. 1993, pp. 3–17.

[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. 1991, pp. 133–138.

[RG17] Guy N. Rothblum and Oded Goldreich. *Simple doubly-efficient interactive proof systems for locally-characterizable sets*. ECCC TR17-018. 2017.

[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. 2016, pp. 49–62.

[RS05] Ran Raz and Amir Shpilka. "Deterministic polynomial identity testing in non-commutative models". In: *Computational Complexity* 14.1 (2005). Preliminary version appeared in CCC '04., pp. 1–19.

[RS96] Ronitt Rubinfeld and Madhu Sudan. "Robust Characterizations of Polynomials with Applications to Program Testing". In: *SIAM Journal on Computing* 25.2 (1996), pp. 252–271.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. "Interactive proofs of proximity: delegating computation in sublinear time". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 793–802.

[Sha92] Adi Shamir. "IP = PSPACE". In: *Journal of the ACM* 39.4 (1992), pp. 869–877.

[She92] Alexander Shen. "IP = PSPACE: Simplified Proof". In: *Journal of the ACM* 39.4 (1992), pp. 878–880.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. "Verifiable Computation with Massively Parallel Interactive Proofs". In: *CoRR* abs/1202.1350 (2012).

[TV07] Luca Trevisan and Salil Vadhan. "Pseudorandomness and Average-Case Complexity Via Uniform Reductions". In: *Computational Complexity* 16.4 (2007), pp. 331–364.

[Tha13] Justin Thaler. "Time-Optimal Interactive Proofs for Circuit Evaluation". In: *Proceedings of the 33rd Annual International Cryptology Conference*. CRYPTO '13. 2013, pp. 71–89.

[Tha15] Justin Thaler. *A Note on the GKR Protocol*. `http://people.cs.georgetown.edu/jthaler/GKRNote.pdf`. 2015.

[WHGSW16] Riad S. Wahby, Max Howald, Siddharth J. Garg, Abhi Shelat, and Michael Walfish. "Verifiable ASICs". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy*. S&P '16. 2016, pp. 759–778.

[WJBSTWW17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. *Full accounting for verifiable outsourcing*. Cryptology ePrint Archive, Report 2017/242. 2017.