Proving as Fast as Computing: Succinct Arguments with Constant Prover Overhead

Noga Ron-Zewi* Ron D. Rothblum†

December 20, 2021

Abstract

Succinct arguments are proof systems that allow a powerful, but untrusted, prover to convince a weak verifier that an input x belongs to a language $L \in \mathsf{NP}$, with communication that is much shorter than the NP witness. Such arguments, which grew out of the theory literature, are now drawing immense interest also in practice, where a key bottleneck that has arisen is the high computational cost of *proving* correctness.

In this work we address this problem by constructing succinct arguments for general computations, expressed as Boolean circuits (of bounded fan-in), with a *strictly linear* size prover. The soundness error of the protocol is an arbitrarily small constant. Prior to this work, succinct arguments were known with a *quasi*-linear size prover for general Boolean circuits or with linear-size only for arithmetic circuits, defined over large finite fields.

In more detail, for every Boolean circuit $C = C(x, w)$, we construct an $O(\log |C|)$-round argument-system in which the prover can be implemented by a size $O(|C|)$ Boolean circuit (given as input both the instance x and the witness w), with arbitrarily small constant soundness error and using $\mathrm{poly}(\lambda, \log |C|)$ communication, where λ denotes the security parameter. The verifier can be implemented by a size $O(|x|) + \mathrm{poly}(\lambda, \log |C|)$ circuit following a size $O(|C|)$ private pre-processing step, or, alternatively, by using a purely public-coin protocol (with no pre-processing) with a size $O(|C|)$ verifier. The protocol can be made zero-knowledge using standard techniques (and with similar parameters). The soundness of our protocol is computational and relies on the existence of collision resistant hash functions that can be computed by linear-size circuits, such as those proposed by Applebaum et al. (ITCS, 2017).

At the heart of our construction is a new information-theoretic *interactive oracle proof* (IOP), an interactive analog of a PCP, for circuit satisfiability, with constant prover overhead. The improved efficiency of our IOP is obtained by bypassing a barrier faced by prior IOP constructions, which needed to (either explicitly or implicitly) encode the entire computation using a multiplication code.

*Department of Computer Science, University of Haifa. Email: noga@cs.haifa.ac.il. Research supported by ISF grant 735/20.

†Department of Computer Science, Technion. Email: rothblum@cs.technion.ac.il. Supported by the Israeli Science Foundation (Grants No. 1262/18 and 2137/19), and the Technion Hiroshi Fujiwara cyber security research center and Israel cyber directorate.

Contents

1 Introduction

A proof system is a method by which a powerful, but untrusted, prover can convince a verifier, which has limited resources, that a computational statement is true. The study of proof systems, ranging from classical NP proofs, to interactive proofs, multi-prover proofs, PCPs and so on, has had a profound impact on several different areas of TCS leading to groundbreaking results such as the theory of NP-completeness, the discovery of zero-knowledge proofs as well as the PCP theorem and its applications to hardness of approximation.

Recently, efficient proof systems have been drawing significant interest also in practice and are now being implemented and deployed.[1] Of particular interest are succinct *argument-systems* [Kil92, Mic00]. These are protocols for proving the correctness of (non-deterministic) computations of size T, so that proving takes time proportional to T, whereas the communication as well as verification time are much smaller than T. Such argument-systems are sound against computationally bounded cheating provers and can often be shown to be *zero-knowledge* meaning that nothing beyond the correctness of the statement is revealed to the verifier.

One of the key efficiency bottlenecks that has been identified for these built systems is the large overhead incurred by the prover in order to prove correctness of the computation. While poly-logarithmic (multiplicative) overhead for the prover has been known for a while (see Section 1.2), the actual cost of proving that a computation is correct is still orders of magnitude slower than merely performing the computation. This fact raises the following fundamental question:

Can proving be as efficient as computing?

In this work we address the above question from a theoretical perspective and construct a succinct argument-system with *constant (multiplicative) overhead* for proving correctness. Our result holds for the model of *Boolean circuits* (with bounded fan-in). That is, we consider computations that are expressed as Boolean circuits and consider the overhead incurred by the prover, when the latter is also implemented as a Boolean circuit.[2]

1.1 Our results

Our main result is an argument-system in which the prover's size is *strictly linear* in the size of the original computation. Soundness holds against any computationally bounded cheating prover, assuming the existence of efficient cryptographic primitives, which have been shown to exist under plausible computational assumptions [AHI+17].

Theorem 1.1 (Informally Stated, see Theorem 3.2)**.** *Assume that there exists a collision-resistant hash function computable by linear-size circuits. Let* $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ *be a size* $S(n,m) \geq n+m$ *Boolean circuit. There exists an* $O(\log(S))$*-round argument-system for the language* $\left\{x \in \{0,1\}^n \ : \ \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ *with arbitrarily small constant soundness error against circuits of size* $\mathrm{poly}(\lambda)$*, where* λ *denotes the security parameter. The prover has size* $O(S)$ *(given the witness as an auxiliary input), and the communication complexity is* $\mathrm{poly}(\log(S), \lambda)$*.*

Furthermore, the verifier can be implemented in size $O(n) + \mathrm{poly}(\log(S), \lambda)$*, following a size* $O(S)$ *private pre-processing step.*

[1] See https://zkproof.org/ and resources therein for further details.

[2] We emphasize that in this regime of parameters the computational model is crucial since simulating a different computational model typically incurs at least a logarithmic overhead.

The protocol of Theorem 3.2 can be applied to circuits composed of arbitrary gates of constant fan-in and fan-out. When measuring the size of the prover, we allow the prover to re-use computations that it performed in prior rounds (in other words the prover is stateful). The protocol can be made zero-knowledge, while retaining the same efficiency parameters, using the standard transformation for public-coin protocols of Ben-Or *et al.* [BGG+88].[3]

We remark that prior to our work, strictly constant overhead was not known even without the furthermore clause of Theorem 1.1 - that is there were no succinct arguments with a linear-size prover for general Boolean circuits even with a polynomial-time verifier (let alone a sub-linear verifier). Rather, the most efficient protocols had poly-logarithmic overhead ([BCGT13], building on [BS08, Kil92] see also [CMT12]). A strictly linear-size prover was known only for arithmetic circuits over large finite fields [BCG+17b, XZZ+19, ZWZZ20, BCG20, BCL20, LSTW21, GLS+21] (see further discussion in Section 1.2).

While the soundness error in Theorem 3.2 is an arbitrarily small constant, it is plausible that a similar result with sub-constant or even negligible soundness could be obtained and this remains a fascinating open question. This is particularly important since protocols with negligible (round-by-round [CCH+19]) soundness error can be made non-interactive using the Fiat-Shamir transform [FS86].

As noted, the verifier of Theorem 1.1 has poly-logarithmic size, following a linear-size pre-processing step (which depends only on the circuit).[4] We emphasize though that the pre-processing step in the protocol is *private* in the sense that the verifier's random coins that were used to produce the state, as well as the state itself, need to be kept hidden from the prover. As an alternative to this private pre-processing, the verifier can be implemented in size linear in the circuit (without any pre-processing) and in this case the protocol is purely public-coin and the communication is still poly-logarithmic.

Remark 1.2. *In most protocols in the literature, the input circuit C is viewed as part of the input for the prover. In our context however even that is too much - the description size of a general circuit of size S is $O(S \cdot \log(S))$, which we cannot afford. Thus, in our construction, rather than receiving the circuit as an input, the prover depends on it explicitly: in particular, the circuit C is hardcoded within the prover's circuit.*

The technical core: Linear-size IOPs. Following a recent line of work [BCS16, AHIV17, BCG+17b, BBHR19, BCR+19, CHM+20, BCG20, BCL20], the improved efficiency in Theorem 3.2 stems from an improvement on an *information-theoretic* proof system. Specifically, we construct an *interactive oracle proof* (IOP) with a linear-size prover. IOPs, introduced by Ben Sasson *et al.* [BCS16] and Reingold *et al.* [RRR16], can be thought of as a hybrid between a classical interactive proof and a PCP. Similarly to an interactive-proof there is a prover and verifier who interact over several rounds. In each round the prover sends a message and the verifier answers with some

[3] Roughly speaking, rather than sending messages in the clear, the prover sends commitments. Since the verifier's messages do not depend on the prover's messages, they can continue the interaction until, at the very end, the prover proves that it knows decommitments that would make the verifier accept using a generic zero-knowledge proof for NP. Since all the original messages are short, and the verifier's decision predicate is small, the total *additive* overhead in the transformation is sub-linear in the circuit size. We remark that this transformation makes a non-blackbox use of the underlying collision-resistant hash function and it would be interesting to obtain a similar result that avoids this (see [BCL20] for such a blackbox-type result for arithmetic circuits over large fields).

[4] A linear dependence is inherent (for general circuits), since a small change in the circuit can totally change its functionality, and would go undetected by a sub-linear time verifier.

random coin tosses. Similarly to a PCP however, the prover messages can be long and the verifier only queries a few bits from each message that it receives. Ben Sasson *et al.* [BCS16] showed how to extend Kilian's PCP based approach [Kil92] for constructing efficient arguments to IOPs. Moreover, as pointed out in [BCG+17b, BCG20], using highly efficient cryptographic primitives (namely, collision resistant hash functions computable by linear-size circuits [AHI+17]), this compilation step preserves the prover efficiency (up to a constant factor).

Thus, our main technical contribution is an IOP for circuit satisfiability with a linear-size prover. In contrast to Theorem 1.1, this result is unconditional and soundness holds even against computationally unbounded cheating provers.

Theorem 1.3 (Informally Stated, see Theorem 3.1)**.** *Let* $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ *be a size* $S(n,m) \geq n+m$ *Boolean circuit. Then, the language* $\left\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ *has a* $O(\log(S))$*-round* IOP *with arbitrarily small constant soundness error. The prover has size* $O(S)$ *(given the witness as an auxiliary input), and the query complexity is* $\mathrm{polylog}(S)$*.*

Furthermore, the verifier can be implemented in size $O(n) + \mathrm{polylog}(S)$*, following a size* $O(S)$ *private pre-processing step.*

Most of the remarks following Theorem 1.1 pertain also to Theorem 1.3. In particular, the prover is stateful and has the circuit C hardwired. As in Theorem 1.1, the pre-processing step of the verifier must be kept hidden from the prover (or alternatively, the verifier can be implemented in size $O(S)$). We assume that the verifier has oracle access to the pre-processed data. Alternatively, since the verifier's queries only depend on its internal randomness and a bound on the size of the circuit, the verifier can perform these queries already in the pre-processing step and save only the answers to these queries.

More on the computational model. As discussed, our focus is on the model of Boolean circuits. The objective of realizing cryptography with linear-size Boolean circuits in general, and zero-knowledge proofs specifically, was explicitly raised by Ishai *et al.* [IKOS08]. We find the model of Boolean circuits to be extremely natural, and an objective and realistic representation of the intrinsic complexity of a computational task. As circuits are a non-uniform model of computation, we take a non-uniform perspective throughout this work.

1.2 Related work

Cryptography with constant computational overhead. The question of constructing general cryptographic primitives with constant computational overhead was raised by Ishai *et al.* [IKOS08], who also constructed several basic cryptographic primitives with such efficiency, including private and public key encryption and semi-honest secure two-party computation. Several more recent works have constructed additional cryptographic primitives with constant overhead. These include universal one-way hash functions [AM17], collision-resistant hash functions [AHI+17] as well as plausibly exponentially secure pseudorandom generators [BIO14] and pseudorandom functions [BIP+18].

Outside (but closely related to) the domain of cryptography, are constructions of error-corrected codes that are linear-time encodable [Spi96, DI14] and decodable [SS96] as well as linear-size pairwise independent hash functions [IKOS08].

We next discuss several works that achieve linear-size provers for arithmetic circuits over large fields, or with large (i.e., at least linear in the circuit size) communication. See also [GLS+21] for a more thorough discussion.

Before proceeding, we emphasize that the focus of this work is theoretical and that there are several works that achieve sub-optimal theoretical bounds but are highly efficient in practice (e.g., [AHIV17]). We do not discuss all of these works here and instead refer the reader to the recent surveys [Ish20, Tha21].

Succinct arguments over large fields. A recent exciting line of work [BCG+17b, XZZ+19, ZWZZ20, BCG20, BCL20, LSTW21, GLS+21] has constructed succinct arguments for *arithmetic* circuits (or more generally, for the R1CS problem described below) over a large finite field, of size that is at least linear in the circuit size, where the prover can be implemented as a linear-size arithmetic circuit. In contrast, our main results apply to *Boolean* circuits. While arithmetic circuits can emulate Boolean circuits, this emulation introduces overhead that is poly-logarithmic in the field size.

Many of the works in the arithmetic setting focus on the NP complete problem of R1CS (for rank-1 constraint satisfaction) rather than circuit satisfiability. Roughly speaking, the input in R1CS consists of matrices A, B and C and a vector x and the question is whether there exists a vector z such that $(Ax') \star (Bx') = Cx'$, where $\star$ represents pointwise multiplication and $x' = (x, z)$. R1CS has a linear-time reduction from circuit satisfiability (where an important benefit is that the circuit can even have large fan-in XOR gates). Prior works considered R1CS over large finite fields, but we believe that our results can be extended also to R1CS over the binary field (in the typical scenario when the matrices are sparse and fixed in advance).

Non-succinct zero-knowledge. A separate line of work has considered constructing zero-knowledge proofs with a linear-size prover, but where the *communication* may also grow linearly in the circuit size (in contrast to succinct arguments which aim for sub-linear length proofs).

Such a non-succinct zero-knowledge proof, with a linear-size prover, can be derived from a basic application of the MPC-in-the-head paradigm of [IKOS09], using commitments computable in linear-size. Similarly to our main results, the resulting protocol only achieves a constant soundness error and achieving sub-constant soundness is open even in this regime (see also [DIK10]). Recent works by Weng *et al.* [WYKW21] and Franzese *et al.* [FKL+21] construct non-succinct zero-knowledge proof (with sub-constant soundness error), where the prover can be implemented as a linear-time RAM program. Concurrent to [WYKW21], and using related techniques, Dittmer *et al.* [DIO21] (see also the followup [YSWW21]) and Baum *et al.* [BMRS21] constructed zero-knowledge proofs for arithmetic circuits with constant overhead.

1.3 Technical overview

As mentioned above, and similarly to recent works on arguments with efficient provers (e.g., [BCG+17b, BCG20, BCL20, LSTW21, GLS+21]), our main contribution is the construction of an efficient IOP (i.e., Theorem 1.3) which can then be transformed in a straightforward manner into an efficient argument using a linear-size collision-resistant hash function [AHI+17], thereby establishing Theorem 1.1. Thus, in this overview we focus on describing the efficient IOP for circuit satisfiability.

Efficient IOPs via multi-sumcheck. The main idea underlying our IOP is the construction of an efficiently encodable systematic[5] code $C : \{0,1\}^n \to \{0,1\}^{O(n)}$ that supports a "multi-sumcheck" protocol with a highly efficient prover. In a *multi-sumcheck* protocol the input is a constant number of codewords $c_1, \ldots, c_d \in C$ and a value $b \in \{0,1\}$ and the goal is to construct an IOP for checking that $\sum_{i \in [n]} c_1(i) \cdots c_d(i) = b$ (where the arithmetic is over the binary field).[6] We wish to construct such a protocol so that the prover can be implemented by a strictly linear-size Boolean circuit and the verifier can be implemented by a poly-logarithimic size circuit, given oracle access to the codewords. The code itself should also be encodable by a linear-size circuit.

The goal in multi-sumcheck should be contrasted with the classical sumcheck protocol of Lund *et al.* [LFKN92].[7] Recall that the latter is a protocol for checking that $\sum_{i \in [n]} c(i) = b$, where c is a codeword from the low degree extension code (i.e., a low degree multivariate polynomial), or more generally any tensor code [Mei13].[8] The classical sumcheck protocol can thus be viewed as an instance of multi-sumcheck with $d = 1$. However, the code used in multi-sumcheck in all prior work is a *multiplication code* — a code in which the pointwise product of any two (or more) codewords results in a codeword from a related code (see details below). Multiplication codes allow for a simple reduction from the case of general d to the case $d = 1$, by simply running the $d = 1$ protocol on the product codeword $c^\star := c_1 \star c_2 \star \cdots \star c_d$, where $a \star b$ denotes the vector obtained by taking the pointwise product of vectors a and b (i.e., $(a \star b)(i) = a(i) \cdot b(i)$, for every coordinate i).

Since there are no known multiplication codes that are encodable by strictly linear-size circuits, in this work we depart from the above paradigm and construct a multi-sumcheck protocol directly for a code C that is *not a multiplication code*. Nevertheless, multiplication codes will play an important role in the construction. We remark that several prior works such as [BCG+17b, RR20, BCG20, BCL20, LSTW21, GLS+21] also do not *explicitly* encode the computation using a multiplication code, but do, implicitly, consider such an encoding, which introduces a super-constant size alphabet and/or (at least) poly-logarithmic overhead.

The use of multi-sumcheck to construct IOPs for circuit satisfiability boils down to what is known as the *arithmetization* step. As it is fairly standard, we postpone an overview of this step (which loosely follows [BCG+17a]) to Section 6.1. It is worth mentioning however that the use of *private* preprocessing by the verifier in our IOP arises because of the arithmetization step. See Section 6.1 for details.

Thus, we skip directly to the overview of the new efficient multi-sumcheck protocol, which is the heart of our construction.

1.3.1 Multi-sumcheck with constant overhead

Since our construction heavily relies on tensor products of codes and multiplication codes, we begin with a brief recap of these objects.

[5] Recall that C is *systematic* if the message appears as the beginning of every codeword.

[6] Here and throughout, for a string $w \in \Sigma^n$ and an index $i \in [n]$, we use $w(i)$ to denote the i-th entry of w.

[7] The sumcheck protocol is typically described as an interactive proof, but interactive proofs are a special case of IOPs (in which the verifier reads all of the bits).

[8] We emphasize that C is a systematic code and so $\sum_{i \in [n]} c(i)$ refers to the summation of the message bits of c.

Tensor codes. Let $D : \{0,1\}^{k_1} \to \{0,1\}^{n_1}$ and $E : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ be linear[9] codes. The *tensor product code* $C : \{0,1\}^{k_1 \times k_2} \to \{0,1\}^{n_1 \times n_2}$, denoted by $C = D \otimes E$, is defined as follows. Given a message $m \in \{0,1\}^{k_1 \times k_2}$ which we view as a $k_1 \times k_2$ dimensional matrix, first we encode each one of the columns of m using the code D and then encode both the original rows, and the new rows generated by D, using the code E. The resulting $n_1 \times n_2$ dimensional matrix is the encoding $C(m)$ of the message m. It is not difficult to see that the rate and minimal relative distance of C are, respectively, equal to the products of the rates and relative distances of D and E.[10] Note that the encoding time T_C of C is equal to $k_2 \cdot T_D(k_1) + n_1 \cdot T_E(k_2)$, where T_D and T_E are the encoding times of D and E, respectively. See Section 2.3.1 for the formal treatment and additional details on tensor codes.

Multiplication codes. Loosely speaking, a linear code C is a *multiplication code*, if the set $\{c \star c' : c, c' \in C\}$ spans a non-trivial error-correcting code $C^\star$. A prime example is the Reed-Solomon code, since the product of two low degree polynomials is still a polynomial (albeit of slightly higher degree). Other notable examples of multiplication codes are the Reed-Muller code, algebraic geometry codes and a construction based on tensor codes [Mei13]. Closely related notions were also studied in the context of secret-sharing and secure multiparty computation (see, e.g., [CDM00, CC06]).

The code C. Recall that our goal is to construct a code C that supports a multi-sumcheck with a linear-size prover and is also encodable in linear-size.

Let $a \geq 1$ be a sufficiently large constant to be determined below. Our code $C = D \otimes E$ is a tensor product of two codes, where $D : \{0,1\}^{n/a} \to \{0,1\}^{O(n/a)}$ and $E : \{0,1\}^{a} \to \{0,1\}^{\mathrm{poly}(a)}$. Notice that this tensor is highly skewed: one dimension is linear in n whereas the other dimension is constant. We assume that D and E are linear and systematic, and therefore C is also linear and systematic.

We turn to our choice of the codes D and E. We choose D to be encodable by linear-size circuits (e.g., Spielman's code [Spi96]), and we get that the encoding time of C is:

$$a \cdot O(n/a) + O(n/a) \cdot T_E(a)$$

which is linear in n, as long as a is constant. We emphasize that we are crucially using the fact that a is constant so that we can use the code E, which is *not* linear-time encodable.

As for E, we choose E to be a *multiplication code*, where the product code $E^\star$ has constant relative distance. The specific choice of the code E is immaterial for this overview, and so we only briefly remark that in the actual construction, since we wish to work over the binary alphabet, and since E can have poor rate, we use the tensor based construction of Meir [Mei13].[11]

We proceed to describe the multi-sumcheck protocol for $C = D \otimes E$. For simplicity, for the rest of this overview we focus on the case $d = 2$, that is, an IOP for checking that $\sum_{i \in [n]} c(i) \cdot c'(i) = b$,

[9] Recall that C is *linear* if it is a linear transformation (over the binary field).

[10] The *rate* of a code $C : \{0,1\}^k \to \{0,1\}^n$ is defined as k/n, and it measures the amount of redundancy in encoding. The *(minimal) relative distance* of C is the minimal *relative Hamming distance* of every two (distinct) codewords, i.e., the minimum fraction of coordinates on which any pair of codewords differ.

[11] In a nutshell, Meir's construction replaces the pointwise product with a tensor product and relies on the observations that the tensor product $c_1 \otimes c_2$ of two codewords $c_1, c_2 \in E$ (1) contains their pointwise product (on the diagonal) and (2) belongs to the tensor code $E^{\otimes 2}$.

where $c, c' \in C$ are two codewords and $b \in \{0,1\}$. The procedure extends in a straightforward manner to any constant number of codewords.

Multi-sumcheck for C: First attempt. Recall that the prover gets as input the two codewords $c, c' \in C = D \otimes E$ and the verifier has oracle access to these codewords. We view c and c' as rectangular matrices of dimension $O(n/a) \times \mathrm{poly}(a)$ in the natural way and index them accordingly: namely, $c(i,j)$ and $c'(i,j)$ refer to the (i,j)-th coordinate of c and c', respectively.

Using the fact that C is a tensor code, we start the construction similarly to the classical sumcheck protocol. Namely, as its first step, in order to prove that $\sum_{i \in [n/a], j \in [a]} c(i,j) \cdot c'(i,j) = b$, the prover sends the message $w \in \{0,1\}^{\mathrm{poly}(a)}$, defined as

$$w = \sum_{i \in [n/a]} c(i,\cdot) \star c'(i,\cdot),$$

where $c(i,\cdot)$ (resp., $c'(i,\cdot)$) denotes the i-th row of the tensor codeword c (resp., c'), the star symbol refers to pointwise multiplication of the rows $c(i,\cdot)$ and $c'(i,\cdot)$, and the sum refers to a sum of codewords belonging to the product code $E^\star$. Note that by linearity, the vector w belongs to the product code $E^\star$.

The message w can be generated in time $(n/a) \cdot \mathrm{poly}(a)$, which is fine since a is a constant. Given w, the verifier first checks that w is a valid codeword of $E^\star$ and that $\sum_{j \in [a]} w(j) = b$. This ensures the verifier that if w was generated correctly then the statement $\sum_{i \in [n/a], j \in [a]} c(i,j) \cdot c'(i,j) = \sum_{j \in [a]} w(j) = b$ is indeed true.

But what if w was generated *incorrectly*? That is, consider a malicious prover that sends $\tilde{w} \neq w$. In this case, since w and $\tilde{w}$ are both codewords of $E^\star$ (this is true for w by definition and true for $\tilde{w}$ since the verifier explicitly checks this condition), and $E^\star$ has constant relative distance, it suffices for the verifier to choose at random $r \in [\mathrm{poly}(a)]$ and check that:

$$\tilde{w}(r) \stackrel{?}{=} w(r) = \sum_{i \in [n/a]} c(i,r) \cdot c'(i,r). \tag{1}$$

It seems that we have made significant progress since Eq. (1) is a new instance of multi-sumcheck, with respect to the codewords $c(\cdot, r), c'(\cdot, r) \in D$, which have dimension that is smaller by a constant factor.

At this point it seems tempting to simply continue by recursion. Indeed, this is the approach taken in the *classical* sumcheck protocol. Unfortunately, this approach runs into a fundamental problem in our setting.

In order to keep recursing we need D to also be a tensor code. As a matter of fact since the recursive step described above only shrinks the problem by a constant factor, we will need to recurse for a super-constant number of steps $t = \omega(1)$. This effectively means that C needs to be a t dimensional tensor of the code E. Denoting the relative distance of E by δ, by the Singleton bound we have that E's rate is at most $1 - \delta$, and therefore the rate of the tensor code $C = E^{\otimes t}$ is $(1-\delta)^t$. Since we need this rate to be a constant (after all C needs to be linear-time encodable), we get that $\delta \approx 1/t$.

However, the soundness error incurred by even just the first round of the protocol is $1 - \delta = 1 - 1/t = 1 - o(1)$, which is already too large.[12] As a matter of fact, things are actually far worse -

[12] To be precise, the soundness error is actually the relative distance of the code $E^\star$ (rather than that of E) but we ignore this minor issue here.

since each round introduces a soundness error of $1-\delta$, the overall soundness error is $1-\delta^t$ which is enormous (as δ is at most a constant and t is logarithmic).

Thus, we cannot assume that D is a tensor code and so it is not clear how to continue the recursion.

How to recurse? Recall that at the end of the first step the prover and verifier have reduced the original multi-sumcheck claim that they had about codewords $c, c' \in C$ to a multi-sumcheck claim about codewords $c(\cdot, r), c'(\cdot, r) \in D$. However, as discussed above, we cannot simply assume that D is a tensor code.

Our approach for resolving this difficulty, which is inspired by the code-switching technique in [RR20], is to have the prover and verifier "switch" the current codewords, which belong to the code D, to equivalent codewords under a new code C'.

At first glance this may seem odd - after all in what way is the code C' better than D? The key observation is that while we use D to encode each one of the $\text{poly}(a)$ columns of c and c', we will only use C' to encode the r-th column, after it is selected. Since we are encoding less information, we can afford for C' to have worse rate than D, and therefore have larger minimal distance.

To facilitate the recursion, the code C' will be similar in spirit to the original code C. Namely, a skewed tensor product of two codes D' and E', but with worse rate and better relative distance than C.

To summarize, at the end of the first iteration the current claim that we have about codewords $c(\cdot, r), c'(\cdot, r) \in D$ is *switched* to a claim on two related codewords, which are encoded under a new code C'. To perform this switch, the prover sends a fresh encoding under the code C' of the messages encoded within $c(\cdot, r)$ and $c'(\cdot, r)$. We remark that these new codewords are quite long (e.g., linear in n) and we rely here on the IOP model which allows the prover to send long messages (from which the verifier will only read a few bits).

For the moment, let us assume that the prover indeed sends the correct new codewords and see in more detail how the recursion proceeds. Later on we will see how we can ensure the consistency of the different encodings.

Gradually reducing the soundness error. Recall that we want to leverage the fact that as the recursion proceeds, the current instance being handled gets smaller and smaller and so we can afford to invest more time on the new instances in order to reduce the (per round) soundness error.

Before describing the parameters in detail, let us first establish the high level picture. In each iteration we reduce the instance size by a factor of a, and so, in iteration ℓ the instance size has reduced to $n_\ell = n/(a^\ell)$. Thus, in each iteration ℓ, we can afford for the prover to run in time $n_\ell \cdot a^{\ell/2} = n/(a^{\ell/2})$ to switch to a code with relative distance $\approx 1 - a^{-\ell/2}$ and therefore reduce the soundness error incurred in this round to (roughly) $a^{-\ell/2}$. Overall this will result in a prover that runs in time $\sum_\ell n/(a^{\ell/2}) = O(n)$ and by the union bound the overall soundness error is $\sum_\ell a^{-\ell/2} \leq 1/2$ (assuming that a is sufficiently large).[13]

With this bird's eye view in mind, let us proceed to a more detailed explanation. Denote the code used in step ℓ of the recursion by $C_\ell = D_\ell \otimes E_\ell$, where we start the indexing at $\ell = 0$ (thus, $C_0 = C$, $C_1 = C'$ and so on). Each code C_ℓ is defined over an alphabet $\mathbb{F}_\ell$ (a finite field) of size

[13] The soundness error can then be reduced to any constant by simply repeating the protocol $O(1)$ times.

roughly $a^{\ell/2}$.[14] We set $D_\ell : \mathbb{F}_\ell^{n/(a^{\ell+1})} \to \mathbb{F}_\ell^{O(n/(a^{\ell+1}))}$ to be encodable in time $(n/a^{\ell+1}) \cdot \mathrm{poly}(a, \ell)$ (concretely we use a straightforward extension of Spielman's code to larger alphabets). We set $E_\ell : \mathbb{F}_\ell^a \to \mathbb{F}_\ell^{a^{\ell/2}}$ to be a multiplication code, encodable[15] in time $a^{\ell/2} \cdot \mathrm{poly}(\ell, a)$, and so that the product code $E_\ell^\star$ has relative distance roughly $\Delta(E_\ell^\star) \approx 1 - a^{-\ell/2}$ (as noted above, in the actual construction, we use the tensor based construction of Meir [Mei13]). This means that each code $C_\ell : \mathbb{F}_\ell^{n/a^\ell} \to \mathbb{F}_\ell^{O(n/a^{\ell/2+1})}$ can be encoded by a circuit of size:

$$a \cdot (n/a^{\ell+1}) \cdot \mathrm{poly}(a, \ell) + (n/a^{\ell+1}) \cdot \mathrm{poly}(a, \ell) \cdot a^{\ell/2} \approx n/(a^{\ell/2}).$$

Summing over all ℓ, this gives us a linear bound on the size of the prover. As for the soundness error, in each round we get a contribution of roughly $1 - \Delta(E_\ell^\star) \approx a^{-\ell/2}$. Therefore, the overall soundness error is bounded by $\sum_\ell a^{-\ell/2}$ which is at most a constant.

To summarize, each round of the protocol starts by the verifier sending a codeword $w_\ell \in E_\ell^\star$ analogous to the codeword w described above. The verifier chooses a random coordinate r_ℓ of w_ℓ to check and the prover in return sends new codewords of the code $C_{\ell+1}$, corresponding to the verifier's choice r_ℓ.

One point worth mentioning is that in later rounds (i.e., when ℓ is about logarithmic in n), checking that $w_\ell \in (E_\ell)^\star$ becomes more challenging, since its codeword length is $a^{\ell/2}$, which can be quite large in the final rounds. One option for resolving this is by making $E_\ell^\star$ locally testable and correctable. However, a simpler solution comes by having the prover send only the *message* encoded within w_ℓ and letting the verifier compute the single point that it is interested in (i.e., the one specified by its random choice) on its own - a task that, for many codes, can be done much more efficiently than computing the entire encoding.

Consistency checking. Only one problem remains - one cannot simply trust the prover to properly re-encode the messages. Thus, we would like a mechanism for checking that a pair of codewords $\alpha \in D_\ell$ and $\beta \in C_{\ell+1}$ are encodings of the exact same message.

To do so we make both D_ℓ and $C_{\ell+1}$ be themselves balanced (i.e., unskewed) constant dimensional tensor codes. For D_ℓ this is straightforward, for example, the code D_0 can be a balanced (constant dimensional) tensor of Spielman's code (which is also linear-time encodable).[16] For $C_{\ell+1}$ this may initially seem odd since we intentionally set it to be a *skewed* tensor (of $D_{\ell+1}$ and $E_{\ell+1}$). However, if both $E_{\ell+1}$ and $D_{\ell+1}$ are themselves balanced tensors, that is $E_{\ell+1} = \left(\widetilde{E}_{\ell+1}\right)^{\otimes t}$ and $D_{\ell+1} = \left(\widetilde{D}_{\ell+1}\right)^{\otimes t}$, then $C_{\ell+1}$ can be simultaneously viewed as a skewed tensor $C_{\ell+1} = E_{\ell+1} \otimes D_{\ell+1}$ and a balanced tensor $C_{\ell+1} = \left(\widetilde{E}_{\ell+1} \otimes \widetilde{D}_{\ell+1}\right)^{\otimes t}$.

Since both D_ℓ and $C_{\ell+1}$'s are balanced tensors, using known techniques (based on [BS06, Vid15, GRR18], see [RR20]) they can be (interactively) *locally tested* and *corrected* (in the relaxed sense of

[14] We increase the field size since we want the code E_ℓ to have high minimal distance, and, by the Plotkin bound [Plo60], the relative distance of a (non-trivial) code with alphabet size q is at most $1 - 1/q$.

[15] For convenience, we use here a code that is encodable in quasi-linear time (in the codeword length), but note that the construction can be easily adapted to any polynomial-time encodable multiplication code.

[16] It may seem alarming at first that we make D_ℓ be a tensor code since the entire point of code switching was to avoid (high-dimensional) tensors. The point is that once we have switched from D_ℓ to $C_{\ell+1}$ we never have to use D_ℓ again, so a constant-dimensional tensor, which we *can* afford, suffices (in particular it suffices for D_ℓ to have constant relative distance).

[GRR18]) with a linear-size prover. These procedures essentially allow us to assume without loss of generality that the messages α and β that the prover sent are valid codewords from the respective codes. However, we still need to ensure that they are encodings of the *exact* same message.

Thus, let us assume that $\alpha \in D_\ell$ and $\beta \in C_{\ell+1}$ are valid codewords and we only need to check that they encode the exact same message. To do so, the verifier chooses a random coordinate λ of α. Since D_ℓ is a systematic linear code, it holds that $\alpha(\lambda) = \sum_i \tau_{i,\lambda} \cdot \alpha(i)$, where the sum is only over the systematic part and $\tau_{i,\lambda}$ are coefficients corresponding to the generating matrix of D_ℓ. Denote by $\hat{\beta}$ the encoding of the message part of β using the code D_ℓ. Note that β and $\hat{\beta}$ *are* encodings of the same message but under different codes.

If the messages encoded in α and β are different, then so are the messages encoded in $\alpha, \hat{\beta} \in D_\ell$. Since D_ℓ has constant relative distance[17] then with constant probability over a random coordinate λ of the codeword α, it holds that:

$$\alpha(\lambda) \neq \hat{\beta}(\lambda) = \sum_i \tau_{i,\lambda} \cdot \hat{\beta}(i) = \sum_i \tau_{i,\lambda} \cdot \beta(i). \tag{2}$$

Equation (2) corresponds to a linear equation over the message of β. Moreover, since the code D_ℓ is a tensor code, the coefficients of this equation correspond to a rank 1 matrix. Thus, we can employ the *sumcheck for tensor codes with rank-1 coefficients* from [RR20] (building on [Mei13]), applied to the codeword β, a codeword of the tensor code $C_{\ell+1}$, to check that the linear constraint of Eq. (2) is indeed satisfied. The prover of this protocol can be implemented in size proportional to the codeword length of $C_{\ell+1}$.

Remark 1.4 (Alphabet Switching via Field Extensions)**.** *The alphabets of the codes D_ℓ and $C_{\ell+1}$ may not be the same. This is because we wanted the alphabet size of the C_ℓ's to gradually increase (in order to reduce the soundness error). We deal with this by ensuring that the alphabet of $C_{\ell+1}$ is a* field extension *of that of D_ℓ. Thus, since the alphabet of D_ℓ is a subfield of that of $C_{\ell+1}$, we can think of the entire consistency protocol as operating over the larger field.*

This concludes the overview of the multi-sumcheck protocol, see the technical sections (i.e., Sections 4 and 5) for additional details. An overview of how multi-sumcheck is used to obtain efficient arguments is provided in Section 6.1 and the detailed proof of this fact is in Sections 6 and 7.

1.4 Organization

Preliminaries are in Section 2. We formally state our results in Section 3. In Section 4 we construct the code family $\mathcal{C}$ that is the pivot of our construction. In Section 5 we give a "multi-sumcheck" protocol for $\mathcal{C}$. In Section 6 we use the multi-sumcheck protocol to construct the efficient IOP and lastly, in Section 7, we transform the latter into an efficient argument-system.

2 Preliminaries

We will often view a string $w \in \Sigma^n$, over an alphabet Σ, as a function $w : [n] \to \Sigma$. In particular, the i-th entry of w is denoted $w(i)$. For strings $x, y \in \Sigma^n$, we let $\mathrm{dist}_\Sigma(x, y)$ denote the fraction

[17] Actually, up to this point we only used the relative distance of the E_ℓ codes and here is the only place where we use the distance of the D_ℓ codes.

of coordinates $i \in [n]$ on which x and y differ, that is, $\mathrm{dist}_\Sigma(x,y) := |\{i \in [n] : x(i) \neq y(i)\}|/n$. Similarly, for $\emptyset \neq S \subseteq \Sigma^n$, we let $\mathrm{dist}_\Sigma(x,S) := \min_{y \in S} \mathrm{dist}_\Sigma(x,y)$. When the alphabet Σ is clear from the context, we omit the subscript Σ.

2.1 Circuits

We next recall the definition of a Boolean circuit. For simplicity we restrict our attention to circuits having only NAND gates with fan-in 2 and fan-out (at most) 2. We also assume that the constants 0 and 1 are always provided as part of the input. Note that circuits as above can emulate circuits with arbitrary gates of constant fan-in and fan-out with constant multiplicative overhead.

Thus, a Boolean circuit is a directed acyclic graph. There are three types of vertices, which we refer to as gates:

- Input gates: having in-degree 0 and out degree 1. The n input gates are associated with distinct integers 1 to n.
- Internal gates: having in-degree 2 and out-degree 1 or 2.
- Output gates: having in-degree 1 and out-degree 0. The output gates are similarly associated with distinct integers 1 to m.

Given a circuit C with n input gates and an input $x \in \{0,1\}^n$ the circuit is evaluated in the following natural manner:

1. For $i \in [n]$, the i-th input gate is labeled by x_i.
2. The label of every internal and output gate is computed recursively by applying the NAND function on the labels of its two children (i.e., the gates that serve as its two inputs).
3. The output of the circuit, denoted $C(x)$, is defined as the label of the output gates.

We say that a circuit C computes the function $f : \{0,1\}^n \to \{0,1\}^m$ if for every $x \in \{0,1\}^n$ it holds that $C(x) = f(x)$.

The size of the circuit C, denoted $|C|$, is defined as the number of vertices in the circuit.

Uniformity. We say that an ensemble of circuits $(C_n)_{n \in \mathbb{N}}$ is T-uniform if there exists a time $T = T(n)$ Turing machine that on input 1^n outputs the adjacency matrix of the graph describing the circuit C_n.

2.2 Interactive oracle proofs

We next define the notion of *interactive oracle proof*, due to [BCS16, RRR16]. We restrict our attention to the public-coin setting which means that all of the verifier's messages simply consist of uniformly random coins. Since we care about very small factors in the parties running times, the definition will be more detailed than usual.

An ℓ-round (public-coin) interactive oracle protocol consists of two entities, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The prover $\mathcal{P}$ consists of ℓ Boolean circuits $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$. For every $i \in [\ell]$, the input to $\mathcal{P}_i$ is the state S_{i-1} from the previous round (where S_0 is simply the main input x and potentially also a witness w) as well as uniformly random coins R_{i-1}, which we think of as being generated

by the verifier (note that R_0 is defined as the empty string). The output of each circuit $\mathcal{P}_i$ is the state S_i for the next round and a message M_i to be transmitted to the verifier. The size $|\mathcal{P}|$ of the prover $\mathcal{P}$ is defined as the sum of the prover circuit sizes, i.e., $|\mathcal{P}| := |\mathcal{P}_1| + \cdots + |\mathcal{P}_\ell|$.

The verifier $\mathcal{V}$ is a Boolean circuit that given as input the transcript $(x, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$ decides whether to accept or reject. We will often be interested in verifiers that run in *sub-linear* time, and in particular are unable to read the entire transcript. In this case we view the input as being separated into two parts $x = (x_{\text{exp}}, x_{\text{imp}})$. The first part, x_{exp} is read explicitly by the verifier (and will often consist of a parameterization of the problem). In contrast, the verifier only has *oracle access* to x_{imp}. We also view the verifier $\mathcal{V}$ as consisting of two separate circuits. The first circuit $\mathcal{V}_1$ takes as input $x_{\text{exp}}, R_1, \ldots, R_{\ell-1}$, and outputs the set of query locations I. The circuit $\mathcal{V}_2$ then gets as input $x_{\text{exp}}, R_1, \ldots, R_{\ell-1}$, as well as the projection of $(x_{\text{imp}}, M_1, \ldots, M_\ell)$ to the query set I, denoted by $(x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$, and based on these decides whether to accept or reject. The size $|\mathcal{V}|$ of the verifier $\mathcal{V}$ is defined as the sum of the sizes of its constituent parts, i.e., $|\mathcal{V}| := |\mathcal{V}_1| + |\mathcal{V}_2|$.

To facilitate composition (see Lemma 2.7 below), it will be useful to assume that $\mathcal{V}_1$ is given in a succinct manner, as defined next.

Definition 2.1. *We say that $\mathcal{V}_1$ is s-*succinct *if there exists a circuit $\hat{\mathcal{V}}_1$ of size s that takes as input $x_{\text{exp}}, R_1, \ldots, R_{\ell-1}$, as well as an index i, and outputs the i-th query location.*

Another property that will be useful for preserving the linear size of the prover during composition is the ability to efficiently project the transcript to the query locations of the verifier.[18]

Definition 2.2. *We say that $\mathcal{V}$ is t-*projectable *if there exists a circuit of size t so that given the transcript $(x = (x_{\text{exp}}, x_{\text{imp}}), M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$, outputs the projection $(x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$ of the transcript to the query locations I.*

Remark 2.3. *The standard definition of interactive oracle protocols in the literature (see e.g., [BCS16, RRR16]) allows the verifier's queries to depend on the entire input x (and sometimes also on answers to previous queries). For sake of simplicity, and to facilitate composition, our definition of interactive oracle protocol only allows the query locations to depend on the explicit input x_{exp} and the verifier's randomness (and implicitly on the input length). We note that many constructions in the literature achieve this stronger notion.*

The key parameters that we will care about are:

1. **Query Complexity:** the number of bits $q = |I|$ that the verifier reads from the input and transcript.
2. **Round complexity:** the number of rounds ℓ.
3. **Verifier Size:** the size of the verifier $\mathcal{V}$, as defined above.

[18] Note that (assuming that $\mathcal{V}_1$ is sufficiently small) every IOP is projectable by a circuit of size $O(cc \cdot q)$, where cc is the transcript length and q denotes the query complexity. This is simply because one can first generate the transcript and query locations and then project to each one of the queries (e.g., using Proposition D.1). However, since we aim for *strictly linear size circuits*, in some cases we will not be able to afford this upper bound.

4. **Prover Size:** the size of the prover $\mathcal{P}$, as defined above. In the context of interactive oracle protocols for NP relations we will often assume that the prover is also given as an auxiliary input a witness w proving that the input x satisfies the relation.

Using the notion of interactive oracle protocols, we can now define interactive oracle proofs.

Definition 2.4 (Interactive oracle proof (IOP))**.** *An ℓ-round interactive oracle proof (IOP) with soundness error ε for a promise problem $(\mathsf{YES}, \mathsf{NO})$ is an ℓ-round (public-coin) interactive oracle protocol $(\mathcal{P}, \mathcal{V})$ such that:*

- **Completeness:** *If $x \in \mathsf{YES}$, then when $\mathcal{V}$ interacts with $\mathcal{P}$, it accepts with probability 1.*
- **Soundness:** *If $x \in \mathsf{NO}$, then for every prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, it accepts with probability at most ε.*

Focusing on *promise problems* allows us to model settings in which the input has some particular structure (e.g., is encoded under an error-correcting code). In particular, this will sometimes allow our verifier to run in time that is *sub-linear* even in the input. Lastly, we note that the standard notion of PCP corresponds to the special case of IOP, when the round complexity is $\ell = 1$.

Private Preprocessing. In this work we will sometimes allow the verifier an initial pre-processing step which is independent of the main input x. We distinguish between public vs. private pre-processing. In *public pre-processing* the random coins that were used to generate the pre-processed state are public, and in particular are known to the prover. In contrast, in *private pre-processing* (which is the type of pre-processing we will be able to achieve), neither the random coins used to generate the pre-processed state nor the state itself are revealed to the prover.

Our focus will be on the private pre-processing model. The model is formally defined by allowing the pre-processed state to depend on the entire randomness used by the verifier in the actual protocol. This randomness is initially hidden from the prover but is then gradually revealed as the protocol progresses.

In any case, we will always separately account for the time (or size) complexity of the pre-processing step.

IOP of Proximity. A particular special case of interest is that of IOP*s of proximity* [BCS16, RRR16], or IOPP for short. For a pair language $\mathcal{L} \subseteq \{(x_{\text{exp}}, x_{\text{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ and $x_{\text{exp}} \in \{0,1\}^*$, we use the notation $\mathcal{L}_{x_{\text{exp}}} := \{x_{\text{imp}} \colon (x_{\text{exp}}, x_{\text{imp}}) \in \mathcal{L}\}$.

Definition 2.5 (Interactive oracle proof of proximity (IOPP))**.** *An ℓ-round IOP of α-proximity (α-IOPP) with soundness error ε for a pair language $\mathcal{L} \subseteq \{(x_{\text{exp}}, x_{\text{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ is an ℓ-round IOP with soundness error ε for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where $\mathsf{YES} = \mathcal{L}$ and $\mathsf{NO} = \{(x_{\text{exp}}, y) : y \text{ is } \alpha\text{-far from } \mathcal{L}_{x_{\text{exp}}}\}$.*

The parameter α is called the proximity parameter. Once more, we note that the standard notion of PCPP corresponds to the special case of IOPP, when the round complexity is $\ell = 1$.

2.2.1 Composition

A key benefit of IOPPs is that they facilitate *composition*. Specifically, one can compose an "outer" *robust* IOP in which the prover has small (e.g., linear) size, but the verifier has relatively large (e.g., slightly sublinear) size, with an "inner" IOPP in which the prover has large (e.g., polynomial) size, but the verifier has small (e.g., logarithmic) size, to obtain the best of both worlds: an IOP with small (e.g., linear) prover size, and small (e.g., logarithmic) verifier size.

Definition 2.6. *We say that an* IOP $(\mathcal{P}, \mathcal{V})$ *for a promise problem* $(\mathsf{YES}, \mathsf{NO})$ *is* (α, ε)-robust *if the soundness requirement is replaced with the following stronger requirement:*

- **Robustness:** *If* $x = (x_{\text{exp}}, x_{\text{imp}}) \in \mathsf{NO}$*, then for every prover strategy* $\mathcal{P}^*$*, when* $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2)$ *interacts with* $\mathcal{P}^*$*, then the answers to the oracle queries that* $\mathcal{V}$ *makes (to* $(x_{\text{imp}}, M_1, \ldots, M_\ell)$*) are* α*-close to making* $\mathcal{V}_2$ *accept, with probability at most* ε*.*

The parameter α is called the Robustness parameter.

Lemma 2.7. *(Composition) Suppose that the following exist:*

- **(Outer IOP:)** *An* ℓ*-round* (α, ε)*-robust* IOP $(\mathcal{P}, \mathcal{V})$ *for a promise problem* $(\mathsf{YES}, \mathsf{NO})$*.*
- **(Inner IOPP:)** *An* ℓ'*-round* q'*-query* α-IOPP $(\mathcal{P}', \mathcal{V}')$ *with soundness error* ε' *for the language*

$$\mathcal{L}_{(\mathcal{P},\mathcal{V})} := \left\{ (x'_{\text{exp}}, x'_{\text{imp}}) \mid \mathcal{V}_2(x'_{\text{exp}}, x'_{\text{imp}}) = \texttt{accept} \right\}.$$

Here $x'_{\text{exp}}, x'_{\text{imp}}$ *are viewed as* $(x_{\text{exp}}, R_1, \ldots, R_{\ell-1})$ *and* $(x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$*, respectively, where* $(x_{\text{exp}}, x_{\text{imp}}, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$ *is a transcript of* $(\mathcal{P}, \mathcal{V})$*, and* I *is the query set of* $\mathcal{V}$ *on this transcript.*

Suppose furthermore that $\mathcal{V}_1$ *is* s*-succinct, and that* $\mathcal{V}$ *is* t*-projectable.*

Then, there exists an $(\ell + \ell')$*-round* q'*-query* IOP $(\mathcal{P}'', \mathcal{V}'')$ *for the promise problem* $(\mathsf{YES}, \mathsf{NO})$ *with soundness error* $\varepsilon + \varepsilon'$*, prover size* $|\mathcal{P}''| = |\mathcal{P}| + |\mathcal{P}'| + t$*, and verifier size* $|\mathcal{V}''_1| \leq |\mathcal{V}'_1| \cdot s$ *and* $|\mathcal{V}''_2| = |\mathcal{V}'_2|$*.*

The use of proof composition originates in [AS98], and is articulated as a composition of a robust PCP with a PCPP in [BGH+06, DR06]. The extension to IOPs is from [BCG+17a] (see also [RR20, Lemma 8.2.] and [ACY21]). As our setting is slightly different (in particular, we care about very small factors in running times), we provide a full proof of Lemma 2.7 in Appendix A.

Lastly, we shall make use of the following PCPP due to [Mie09] as the "inner PCPP" in our composition steps.

Theorem 2.8 ([Mie09, Theorem 1])**.** *Let* $\mathcal{L}$ *be a pair language decidable in time* $T = T(m + n)$*, where* m, n *are the explicit and implicit input lengths, respectively. Then for any* $\alpha, \varepsilon > 0$*, there exists a* $\text{poly}(\log(1/\varepsilon)/\alpha)$*-query* α-PCPP *for* $\mathcal{L}$ *with soundness error* ε*, prover's running time* $\text{poly}(m, n, T(m + n))$*, and verifier's running time* $\text{poly}(m, \log n, \log(T(m + n)), \log(1/\varepsilon)/\alpha)$*.*

Remark 2.9. *We remark on the following changes in the above theorem, compared to Theorem 1 of [Mie09]:*

1. *Theorem 1 of [Mie09] does not explicitly state the prover's running time but it can be readily verified that the prover can be implemented in the stated time bound.*

2. *Theorem 1 of [Mie09] is only stated for constant proximity parameter and contant soundness error. However, in [RR20, Theorem 8.6] it is shown how to extend this result to arbitrary small proximity parameter α and soundness error ε, at the cost of increasing the verifier's running time and query complexity by a multiplicative factor of* $\mathrm{poly}(\log(1/\varepsilon)/\alpha)$.

Instantiating the inner IOPP in Lemma 2.7 with the PCPP given in Theorem 2.8, readily implies the following useful corollary.

Corollary 2.10. *Let* $(\mathsf{YES}, \mathsf{NO})$ *be a promise problem, where* m, n *are the explicit and implicit input lengths, respectively. Suppose that there exists an* ℓ*-round* (α, ε)*-robust* IOP $(\mathcal{P}, \mathcal{V})$ *for a promise problem* $(\mathsf{YES}, \mathsf{NO})$*, satisfying that:*

1. $(\mathcal{P}, \mathcal{V})$ *has query complexity* q *and randomness complexity* r.
2. *The language* $\mathcal{L}_{(\mathcal{P},\mathcal{V})}$*, as defined in Lemma 2.7, can be decided in time* T.
3. $\mathcal{V}_1$ *is* s*-succinct, and* $\mathcal{V}$ *is* t*-projectable.*

Then for any $\varepsilon' > 0$*, there exists an* $(\ell+1)$*-round* $\mathrm{poly}(\log(1/\varepsilon')/\alpha)$*-query* IOP $(\mathcal{P}', \mathcal{V}')$ *for the promise problem* $(\mathsf{YES}, \mathsf{NO})$ *with soundness error* $\varepsilon + \varepsilon'$*, prover size* $|\mathcal{P}'| = |\mathcal{P}| + t + \mathrm{poly}(m, q, r, T)$*, and verifier size* $|\mathcal{V}'| = \mathrm{poly}\left(m, r, \log q, \log(T), s, \log(1/\varepsilon')/\alpha\right)$.

2.3 Error-correcting codes

Let Σ be a finite alphabet, and k, n be positive integers (the message length and the codeword length, respectively). An (error-correcting) code is an injective map $C : \Sigma^k \to \Sigma^n$. The elements in the domain of C are called messages, and the elements in the image of C are called codewords. We say that C is systematic if the message is a prefix of the corresponding codeword, i.e., for every $x \in \Sigma^k$ there exists $z \in \Sigma^{n-k}$ such that $C(x) = (x, z)$.

The rate of a code $C : \Sigma^k \to \Sigma^n$ is the ratio $\rho := \frac{k}{n}$. The relative distance $\mathrm{dist}(C)$ of C is the maximum $\delta > 0$ such that for every pair of distinct messages $x, y \in \Sigma^k$ it holds that $\mathrm{dist}_\Sigma(C(x), C(y)) \geq \delta$.

If $\Sigma = \mathbb{F}$ for some finite field $\mathbb{F}$, and C is a linear map between the vector spaces $\mathbb{F}^k$ and $\mathbb{F}^n$ then we say that C is linear. The generating matrix of a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ is a matrix $G \in \mathbb{F}^{n \times k}$ such that $C(x) = G \cdot x$ for any $x \in \mathbb{F}^k$.

2.3.1 Tensor codes

A main ingredient in our constructions is the tensor product operation, defined as follows (see, e.g., [Sud01, DSW06]).

Definition 2.11 (Tensor codes)**.** *The* tensor product code *of linear codes* $C : \mathbb{F}^k \to \mathbb{F}^n$ *and* $C' : \mathbb{F}^{k'} \to \mathbb{F}^{n'}$ *is the code* $C \otimes C' : \mathbb{F}^{k \times k'} \to \mathbb{F}^{n \times n'}$*, where the encoding* $(C \otimes C')(M)$ *of any message* $M \in \mathbb{F}^{k \times k'}$ *is obtained by first encoding each column of* M *with the code* C*, and then encoding each resulting row with the code* C'.

Note that by linearity, the codewords of $C \otimes C'$ are $n \times n'$ matrices (over the field $\mathbb{F}$) whose columns belong to the code C, and whose rows belong to the code C'. It is also known that the converse is true: any $n \times n'$ matrix, whose columns belong to the code C, and whose rows belong to the code C', is a codeword of $C \otimes C'$.

Fact 2.12. *A matrix $w \in \mathbb{F}^{n \times n'}$ is a codeword of $C \otimes C'$ if and only if the restriction of w to any column is a codeword of C, and the restriction of w to any row is a codeword of C'.*

The following effects of the tensor product operation on the classical parameters of the code are well known.

Fact 2.13. *Suppose that $C : \mathbb{F}^k \to \mathbb{F}^n$, $C' : \mathbb{F}^{k'} \to \mathbb{F}^{n'}$ are linear codes of rates ρ, ρ' and relative distances δ, δ' respectively. Then, the tensor product code $C \otimes C'$ is a linear code of rate $\rho \cdot \rho'$ and relative distance $\delta \cdot \delta'$.*

By the definition of the tensor product code, we also have the following.

Claim 2.14. *The following holds for any pair of linear codes $C : \mathbb{F}^k \to \mathbb{F}^n$, $C' : \mathbb{F}^{k'} \to \mathbb{F}^{n'}$.*

1. *If C, C' can be encoded by a T, T'-uniform Boolean circuits of sizes s, s' respectively, then $C \otimes C'$ can be encoded by a $(T + T' + n' \cdot s + n \cdot s')$-uniform Boolean circuit of size $n' \cdot s + n \cdot s'$.*
2. *If C, C' can be encoded in time T, T', respectively, then $C \otimes C'$ can be encoded in time $n' \cdot T + n \cdot T'$.*
3. *If each coordinate of C, C' can be computed in time T_0, T'_0, respectively, then each coordinate of $C \otimes C'$ can be computed in time $k' \cdot T_0 + T'_0$.*

For a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$, let $C^{\otimes 1} := C$ and $C^{\otimes t} := C \otimes C^{\otimes (t-1)}$, for any $t \geq 2$. As in the 2-dimensional case, the codewords of $C^{\otimes t} : \mathbb{F}^{k^t} \to \mathbb{F}^{n^t}$ can be viewed as t-dimensional cubes, satisfying that their projection on any axis-parallel line is a codeword of C. Once more, we have that the converse is also true.

Fact 2.15. *A cube $w \in \mathbb{F}^{n^t}$ is a codeword of $C^{\otimes t}$ if and only if the restriction of w to any axis-parallel line is a codeword of C.*

Moreover, by Fact 2.13 if C is a linear code of rate ρ and relative distance δ then $C^{\otimes t}$ is a linear code of rate ρ^t and relative distance δ^t. Finally, applying iteratively the above Claim 2.14, gives the following.

Claim 2.16. *The following holds for any linear code $C : \mathbb{F}^k \to \mathbb{F}^n$.*

1. *If $C : \mathbb{F}^k \to \mathbb{F}^n$ can be encoded by a Boolean circuit of size s, then $C^{\otimes t}$ can be encoded by a Boolean circuit of size $tn^{t-1}s$.*
2. *If $C : \mathbb{F}^k \to \mathbb{F}^n$ can be encoded in time T, then $C^{\otimes t}$ can be encoded in time $tn^{t-1}T$.*
3. *If each coordinate of C can be computed in time T_0, then each coordinate of $C^{\otimes t}$ can be computed in time $O(k^{t-1}T_0)$.*

2.3.2 Constructible finite fields

In this work we will use finite fields of varying sizes. For sake of efficient implementation of the field operations, we need the field to be *constructible*, in the following sense.

Definition 2.17. *We say that an ensemble of finite fields $\mathbb{F} = (\mathbb{F}_n)_{n\in\mathbb{N}}$ is* constructible *if elements in $\mathbb{F}_n$ can be represented by $O(\log(|\mathbb{F}_n|))$ bits and field operations (i.e., addition, subtraction, multiplication, inversion and sampling random elements) can all be performed in $\mathrm{polylog}(|\mathbb{F}_n|)$ time given this representation.*

Lemma 2.18 (see [Sho88])**.** *For every $S = S(n) \geq 1$, there exists a constructible field ensemble of characteristic 2 and size $O(S)$.*

For a constructible finite field of characteristic 2, we shall assume that all elements in $\mathbb{F}$ are represented as codewords of a binary error-correcting code of message length $\log(|\mathbb{F}|)$ and constant rate and relative distance, that is efficiently (poly-time) encodable and decodable. Note that any constructible finite field can be turned into this representation while preserving the property of being constructible.

2.3.3 Specific families of codes

For our code construction we shall use as a base code in the tensor code construction two specific families of codes. The first of these is the well-known family of Reed-Solomon codes.

Fact 2.19 (Reed-Solomon codes, [RS60])**.** *For any constructible finite field $\mathbb{F}$ and integers $k \leq n \leq |\mathbb{F}|$, there exists a systematic linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ of relative distance at least $1 - \frac{k}{n}$ that is encodable in time $\mathrm{poly}(n, \log(|\mathbb{F}|))$. Moreover, each coordinate of C can be computed in time $\mathrm{poly}(k, \log(|\mathbb{F}|))$.*

We shall also use the linear-time encodable codes of [Spi96]. While the original construction is over the binary field, it is easily adaptable to any constructible finite field $\mathbb{F}$ by simply making all computations over $\mathbb{F}$.

Theorem 2.20 (Linear-time encodable codes [Spi96], Theorem 19)**.** *For any constructible finite field $\mathbb{F}$, there exists a code family $\{C_k\}_{k\in\mathbb{N}}$, where $C_k : \mathbb{F}^k \to \mathbb{F}^n$ is a systematic linear code of constant rate and constant relative distance that is encodable by a $\mathrm{poly}(s)$-uniform Boolean circuit of size $s := k \cdot \mathrm{polylog}(|\mathbb{F}|)$.*

2.4 Evading set

Loosely speaking, an *ε-evading set* A is a relatively small set such that for any non-zero test-vector y, the inner product of y with a random $x \leftarrow A$ is non-zero with probability at least ε. Evading sets are very similar to *small bias sets* of Naor and Naor [NN93], where the only difference is in terms of parameters - while for small bias sets the inner product is non-zero with probability close to $1/2$, for an evading set this probability need only be some small constant that is greater than 0.

Definition 2.21 (Evading Set)**.** *A function $S : \{0,1\}^d \to \{0,1\}^n$ is an ε-*evading set generator *if for every $0 \neq y \in \{0,1\}^n$ it holds that $\Pr_{x\leftarrow\{0,1\}^d}[\langle S(x), y\rangle \neq 0] \geq \varepsilon$.*

Above, the inner product operation is defined over the field $\mathbb{GF}(2)$ (i.e., $\langle x, y\rangle = \sum_i x(i) \cdot y(i) \bmod 2$).

We shall need the following proposition, which shows the existence of a linear-size evading set generator. The proof is deferred to Appendix E.

Proposition 2.22. *(Linear-size evading set generator) There exists $d = O(\log n)$, and an 0.49-evading set generator $S : \{0,1\}^d \to \{0,1\}^n$ that can be evaluated by a size $O(n)$ Boolean circuit.*

3 Our results

In this section we formally state our main results. The first main result, is an IOP for circuit satisfiability, with a linear-size prover. This result is unconditional.

Theorem 3.1. *Let* $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ *be a size* $S(n,m) \geq n+m$ *Boolean circuit. There exists an* $O(\log S)$*-round* IOP *for the language* $\left\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ *with constant soundness error. The prover, given as auxiliary input also the witness* w*, can be implemented by a size* $O(S)$ *Boolean circuit. After a size* $O(S)$ *private pre-processing step, the verifier can be implemented by a size* $O(n) + \mathrm{polylog}(S)$ *Boolean circuit, and has* $\mathrm{polylog}(S)$ *query complexity.*

The second main result is an argument-system for circuit satisfiability with a linear-size prover. We assume here the existence of a linear-size computable hash function and the soundness holds against computationally bounded cheating provers.

Theorem 3.2. *Assume that there exists a collision-resistant hash function computable by linear size circuits, with shrinkage factor* $1/2$*. Let* $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ *be a size* $S(n,m) \geq n+m$ *Boolean circuit. There exists an* $O(\log S)$*-round argument-system for the language* $\left\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ *with constant soundness error against circuits of size* $\mathrm{poly}(\lambda)$*, where* $\mathrm{poly}(\lambda) \leq S$ *denotes the security parameter. The prover, given as auxiliary input also the witness* w*, can be implemented by a size* $O(S)$ *Boolean circuit. After a size* $O(S)$ *private pre-processing step, the verifier can be implemented by a size* $O(n) + \mathrm{poly}(\log(S), \lambda)$ *Boolean circuit. The total communication complexity is* $\mathrm{poly}(\log(S), \lambda)$*.*

4 The code family $\mathcal{C}$ and main lemmas

4.1 The code family $\mathcal{C}$

We first describe the code family $\mathcal{C} = \{C_\ell\}_\ell$ which we use for our multi-sumcheck protocol. We start by fixing some parameters. Fix an integer $d \geq 2$, and let $t \geq 2$ be a sufficiently large integer, to be determined later on.[19] Let q be the smallest power of 2 that is larger than $(4td)^t$, and let $a := q^{t-1}$.

Sequence of finite fields. Next we define a sequence of finite fields $\mathbb{F}_0, \mathbb{F}_1, \ldots$, where the code C_ℓ will be a linear code over $\mathbb{F}_\ell$. Let $\mathbb{F}_0$ be the binary field (i.e., $|\mathbb{F}_0| = 2$). For every $\ell = 1, 2, \ldots$, let $\mathbb{F}_\ell$ be the smallest finite field of order $|\mathbb{F}_\ell| \geq q^\ell$ such that $|\mathbb{F}_\ell| = q^{2^z}$ for some integer $z \geq 1$.[20] Note that $|\mathbb{F}_\ell| \leq q^{2\ell}$ for every $\ell \in \mathbb{N}$ (since there must be some power of 2 between ℓ and 2ℓ). Observe that each $\mathbb{F}_\ell$ is a field extension of $\mathbb{F}_{\ell-1}$ (this includes the case that $\mathbb{F}_\ell = \mathbb{F}_{\ell-1}$). We further assume that $\mathbb{F}_\ell$ is constructible and that the elements of $\mathbb{F}_\ell$ are represented by an efficiently encodable binary error-correcting code of constant relative distance (in the sense of Section 2.3.2).[21]

[19] Jumping ahead, d denotes the number of codewords for the multi-sumcheck protocol, and t is the dimension of the tensor codes involved in the construction. The reader may find it useful to simply think of $d = 2$ and $t = 1$, although in the actual construction they will need to be larger.

[20] The reason that we set $|\mathbb{F}_0| = 2$ (rather than the more natural choice of $|\mathbb{F}_0| = q$) is that we want the code C_0 to be a binary code.

[21] In a nutshell, this representation allows us to directly translate robustness (cf., Definition 2.6) over the large alphabet $\mathbb{F}_\ell$ to robustness over the binary alphabet (with only a constant factor degradation).

To define the code family $\mathcal{C} = \{C_\ell\}_\ell$, we first define two other code families $\mathcal{D} = \{D_\ell\}_\ell$ and $\mathcal{E} = \{E_\ell\}_\ell$, and then for any ℓ, we set $C_\ell = D_\ell \otimes E_\ell$.

The code family $\mathcal{D}$. For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, we choose $\widetilde{D}_\ell$ to be the linear-time encodable code over $\mathbb{F}_\ell$ of message length $\left(\frac{n}{a^{\ell+1}}\right)^{1/t}$ guaranteed by Theorem 2.20, and we let $D_\ell = (\widetilde{D}_\ell)^{\otimes t}$. By Theorem 2.20 and the properties of tensor codes (cf., Section 2.3.1) we have the following.

Proposition 4.1 (Properties of $\widetilde{\mathcal{D}}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code $\widetilde{D}_\ell$ satisfies the following properties:*

1. *$\widetilde{D}_\ell$ is a systematic linear code over $\mathbb{F}_\ell$ of message length $\left(\frac{n}{a^{\ell+1}}\right)^{1/t}$ and codeword length $\Theta\left(\left(\frac{n}{a^{\ell+1}}\right)^{1/t}\right)$.*
2. *$\widetilde{D}_\ell$ has relative distance at least δ_0 for some absolute constant $\delta_0 > 0$.*
3. *$\widetilde{D}_\ell$ can be encoded by a $\mathrm{poly}(s)$-uniform circuit of size $s := \left(\frac{n}{a^{\ell+1}}\right)^{1/t} \cdot \mathrm{polylog}(|\mathbb{F}_\ell|)$.*

Proposition 4.2 (Properties of $\mathcal{D}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code D_ℓ satisfies the following properties:*

1. *D_ℓ is a systematic linear code over $\mathbb{F}_\ell$ of message length $\frac{n}{a^{\ell+1}}$ and codeword length $2^{\Theta(t)} \cdot \frac{n}{a^{\ell+1}}$.*
2. *D_ℓ has relative distance at least $(\delta_0)^t$ for some absolute constant $\delta_0 > 0$.*
3. *D_ℓ is a t-dimensional tensor product.*
4. *D_ℓ can be encoded by a Boolean circuit of size $s := \frac{n}{a^{\ell+1}} \cdot \mathrm{poly}(\log(|\mathbb{F}_\ell|), 2^t)$.*

The code family $\mathcal{E}$. For $\ell = 0$, we choose $\widetilde{E}_0$ to be any explicit asymptotically good binary linear code of message length $a^{1/t}$, for example the one guaranteed by Theorem 2.20. For any $\ell \geq 1$, we choose $\widetilde{E}_\ell$ to be a Reed-Solomon code over $\mathbb{F}_\ell$ of message length $a^{1/t}$ and codeword length $(a \cdot |\mathbb{F}_\ell|)^{1/t}$. Note that by our choice of $a = q^{t-1}$ we have that $(a \cdot |\mathbb{F}_\ell|)^{1/t} = q^{(t-1)/t} \cdot |\mathbb{F}_\ell|^{1/t} \leq |\mathbb{F}_\ell|$, and so such a code exists. Finally, we let $E_\ell = (\widetilde{E}_\ell)^{\otimes t}$.[22]

By the properties of Reed-Solomon codes (cf., Fact 2.19) and tensor codes (cf., Section 2.3.1) we have the following.

Proposition 4.3 (Properties of $\widetilde{\mathcal{E}}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code $\widetilde{E}_\ell$ satisfies the following properties:*

1. *$\widetilde{E}_\ell$ is a systematic linear code over $\mathbb{F}_\ell$ of message length $a^{1/t}$ and codeword length $\Theta\left((a \cdot |\mathbb{F}_\ell|)^{1/t}\right)$.*
2. *For any $\ell \geq 1$ the code $\widetilde{E}_\ell$ has relative distance at least $1 - \frac{1}{|\mathbb{F}_\ell|^{1/t}}$, and the code E_0 has relative distance at least δ_0 for some absolute constant $\delta_0 > 0$.*

[22] The reason that E_0 is different from the other codes in the family is again due to the fact that we want C_0 to be a *binary* code (c.f., Footnote 20). We also note a mild difference between the definition here and the description in the overview in Section 1.3: In the overview the codes $\{E_\ell\}_\ell$ were set to be multiplication codes whereas here we do not explicitly rely on this fact. Rather, we implicitly use here the multiplication code construction of Meir [Mei13] based on tensoring. See further discussion on beginning of Section 5.2.

3. *$\widetilde{E}_\ell$ can be encoded in time* $\mathrm{poly}(a^{1/t}, |\mathbb{F}_\ell|^{1/t}, \log(|\mathbb{F}_\ell|))$*, and each coordinate of $\widetilde{E}_\ell$ can be computed in time* $\mathrm{poly}(a^{1/t}, \log(|\mathbb{F}_\ell|))$.

Proposition 4.4 (Properties of $\mathcal{E}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code E_ℓ satisfies the following properties:*

1. *E_ℓ is a systematic linear code over $\mathbb{F}_\ell$ of message length a and codeword length $2^{\Theta(t)} \cdot a \cdot |\mathbb{F}_\ell|$*
2. *For any $\ell \geq 1$ the code E_ℓ has relative distance at least $\left(1 - \frac{1}{|\mathbb{F}_\ell|^{1/t}}\right)^t$, and the code E_0 has relative distance at least $(\delta_0)^t$ for some absolute constant $\delta_0 > 0$.*
3. *E_ℓ is a t-dimensional tensor product.*
4. *E_ℓ can be encoded in time* $\mathrm{poly}(a, |\mathbb{F}_\ell|, 2^t)$*, and each coordinate of E_ℓ can be computed in time* $\mathrm{poly}(a, \log(|\mathbb{F}_\ell|))$.

The code family $\mathcal{C}$. For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, we define the code $\widetilde{C}_\ell$ as the tensor product $\widetilde{C}_\ell = \widetilde{D}_\ell \otimes \widetilde{E}_\ell$, and we let $C_\ell = (\widetilde{C}_\ell)^{\otimes t} = D_\ell \otimes E_\ell$. By the properties of tensor codes we have the following.

Proposition 4.5 (Properties of $\widetilde{\mathcal{C}}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code $\widetilde{C}_\ell$ satisfies the following properties:*

1. *$\widetilde{C}_\ell$ is a systematic linear code over $\mathbb{F}_\ell$ of message length $\left(\frac{n}{a^\ell}\right)^{1/t}$ and codeword length $\Theta\left(\left(\frac{n}{a^\ell} \cdot |\mathbb{F}_\ell|\right)^{1/t}\right)$.*
2. *$\widetilde{C}_\ell$ has relative distance at least δ_0 for some absolute constant $\delta_0 > 0$.*
3. *$\widetilde{C}_\ell$ can be encoded by a* $\mathrm{poly}(s)$*-uniform circuit of size $s := \left(\frac{n}{a^\ell}\right)^{1/t} \cdot \mathrm{poly}(a^{1/t}, |\mathbb{F}_\ell|^{1/t}, \log(|\mathbb{F}_\ell|))$.*

Proposition 4.6 (Properties of $\mathcal{C}$)**.** *For any $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, the code C_ℓ satisfies the following properties:*

1. *C_ℓ is a systematic linear code over $\mathbb{F}_\ell$ of message length $\frac{n}{a^\ell}$ and codeword length $2^{\Theta(t)} \cdot \frac{n}{a^\ell} \cdot |\mathbb{F}_\ell|$.*
2. *C_ℓ has relative distance at least $(\delta_0)^t$ for some absolute constant $\delta_0 > 0$.*
3. *C_ℓ is a t-dimensional tensor product.*
4. *C_ℓ can be encoded by a Boolean circuit of size $s := \frac{n}{a^\ell} \cdot \mathrm{poly}(a, |\mathbb{F}_\ell|, 2^t)$.*

4.2 $\mathcal{C}$-encoded IOPs

Throughout the technical sections, it will be convient for us to restrict our attention to IOPs in which both the honest, and potentially cheating, provers are only allowed to send messages that belong to the code family $\mathcal{C}$. We refer to such a restricted IOP (which is defined formally below) as a *$\mathcal{C}$-encoded* IOP. This notion is similar in spirit to "IOPs with encoded provers" [RRR16] and "polynomial IOPs" [BFS20, CHM+20] which consider IOPs in which the honest and dishonest provers are only allowed to send messages that correspond to low degree polynomials. Also, similarly to the case of polynomial IOPs, we mention that using the local testability and (relaxed) local

correction properties of $\mathcal{C}$, it is straightforward to convert a $\mathcal{C}$-encoded IOP into a standard IOP (see Proposition 4.7 below).

In more detail, a $\mathcal{C}$-encoded IOP $(\mathcal{P}, \mathcal{V})$ is defined exactly like a standard IOP (see Definition 2.4) with the following modifications. As part of the specification of the verifier $\mathcal{V}$, each round i is associated with one of the codes $C_{\ell(i)}$ in the code family. We say that a prover strategy $\mathcal{P}^*$ is $\mathcal{C}$-encoded if in every round i, the prover always sends a codeword from $C_{\ell(i)}$. We require that the honest prover $\mathcal{P}$ is $\mathcal{C}$-encoded and relax the soundness condition to hold only with respect to every $\mathcal{C}$-encoded cheating prover $\mathcal{P}^*$ (rather than all possible cheating prover strategies).

For convenience we also include the "identity code" $I(m) = m$ in the code family $\mathcal{C}$. This is done to allow the prover in our IOP to also send plain messages when it needs to. We emphasize that this does *not* let the cheating prover send un-encoded messages whenever it wants, because the specification of the index of the code sent in every round is included as part of the description of the protocol (and is therefore known to the verifier).

The following proposition shows how to transform a $\mathcal{C}$-encoded IOP into a standard IOP.

Proposition 4.7. *Suppose that $\mathcal{L}$ has an ℓ-round $\mathcal{C}$-encoded IOP $(\mathcal{P}, \mathcal{V})$ with constant soundness error. Suppose furthermore that the verifier makes a constant number of queries to the input, and for each message of $\mathcal{P}$, the verifier either reads the entire message, or makes a constant number of queries to this message.*

Then $\mathcal{L}$ also has an $(\ell + 2)$-round standard IOP $(\mathcal{P}', \mathcal{V}')$ with constant soundness error. If the prover and verifier in $(\mathcal{P}, \mathcal{V})$ have sizes $|\mathcal{P}|, |\mathcal{V}|$, respectively, then the prover and verifier in $(\mathcal{P}', \mathcal{V}')$ have sizes $O(|\mathcal{P}|)$ and $\mathrm{poly}(|\mathcal{V}|) + \ell \cdot \mathrm{polylog}(|\mathcal{P}|)$, respectively. Moreover, the verifier makes a constant number of queries to the input, and for each message of $\mathcal{P}'$, the verifier $\mathcal{V}'$ either reads the entire message, or makes a constant number of queries to this message.

The proof of Proposition 4.7 capitalizes on the fact that each of the codes in $\mathcal{C}$ is a *tensor code* and is by now routine. We therefore defer the proof to Appendix B.

4.3 Multi-sumcheck

The main technical result that we rely on is that the code C_0 supports a multi-sumcheck protocol with a linear-size prover. This fact is captured by the following key lemma.

Lemma 4.8 (Multi-sumcheck with constant overhead)**.** *Let $d \in \mathbb{N}$ be a constant, and consider the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \Big\{(c_1, \ldots, c_d, b) : \sum_{i \in [n]} c_1(i) \cdots c_d(i) = b\Big\},$$
$$\mathsf{NO} = \Big\{(c_1, \ldots, c_d, b) : \sum_{i \in [n]} c_1(i) \cdots c_d(i) \neq b\Big\},$$

where $c_1, \ldots, c_d \in C_0$, $b \in \{0, 1\}$, and n denotes the message length of C_0. There exists an $O(\log n)$-round $\mathcal{C}$-encoded IOP for $(\mathsf{YES}, \mathsf{NO})$ with constant soundness error. The prover has size $O(n)$, and the verifier has size $\mathrm{polylog}(n)$.

Furthermore, the verifier makes a constant number of queries to the input, and for each prover message, the verifier either reads the entire message, or makes a constant number of queries to this message.

5 Multi-sumcheck with constant overhead

In this section we construct the multi-sumcheck protocol with constant overhead, thus proving Lemma 4.8. For this, we first introduce in Section 5.1 below a *consistency checking protocol* for checking that two codewords of two different tensor codes are encodings of the same message. This protocol will then be used as a sub-procedure in the multi-sumcheck protocol given in Section 5.2.

5.1 Consistency checking

In this section we show a protocol for checking that two codewords, c and c', of two (possibly different) t-dimensional tensor codes are encodings of the exact same message. The tensor codes may have different codeword lengths, and may be defined over different fields, as long as the message lengths are the same, and one of the fields is an extension field of the other.

Lemma 5.1 (Consistency checking)**.** *Let $\mathbb{F}$ and $\mathbb{F}'$ be constructible finite fields of characteristic 2, where $\mathbb{F}'$ is an extension field of $\mathbb{F}$.*[23] *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ and $C' : (\mathbb{F}')^k \to (\mathbb{F}')^{n'}$ be systematic linear codes of relative distance δ, δ', respectively. Then for every integer $t \geq 1$, there exists a $(t + O(1))$-round $\mathrm{poly}(t/\delta)$-query IOP $(\mathcal{P}, \mathcal{V})$ with soundness error $1 - \frac{1}{2} \cdot (\delta \cdot \delta')^t$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \left\{ \left(C^{\otimes t}(x), (C')^{\otimes t}(x')\right) \ : \ x = x' \right\},$$
$$\mathsf{NO} = \left\{ \left(C^{\otimes t}(x), (C')^{\otimes t}(x')\right) \ : \ x \neq x' \right\}.$$

Assuming that each coordinate of C can be computed by a $\mathrm{poly}(T)$-uniform circuit of size T, and that C' can be encoded in time T', the prover has size $O(t \cdot (n')^{t-1} \cdot (T' + T)) + \mathrm{poly}(T, T', t)$, and the verifier has size $\mathrm{poly}(\log(T), \log(T'), t/\delta)$.

The proof of the above lemma relies on the following *sumcheck protocol for rank 1 tensor coefficients* from [RR20]. This protocol allows a verifier, who is given oracle access to a tensor encoding of x, to verify expressions of the form $\langle \lambda, x \rangle = b$, where λ is a rank 1 tensor (Namely, there exist $\lambda_1, \lambda_2, \ldots, \lambda_t \in \mathbb{F}^n$ so that $\lambda = \lambda_1 \otimes \lambda_2 \otimes \cdots \otimes \lambda_t \in \mathbb{F}^{n^t}$, where $(\lambda_1 \otimes \lambda_2 \otimes \cdots \otimes \lambda_t)(\bar{i}) = \lambda_1(i_1) \cdot \lambda_2(i_2) \cdots \lambda_t(i_t)$ for any $\bar{i} = (i_1, i_2, \ldots, i_t) \in [n]^t$).

Lemma 5.2. *(Sumcheck for rank 1 tensor coefficients) Let $\mathbb{F}$ be a constructible finite field of characteristic 2, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 1$, there exists a $(t + 1)$-round $\mathrm{poly}(t/\delta)$-query IOP $(\mathcal{P}, \mathcal{V})$ with soundness error $1 - \frac{1}{2} \cdot \delta^t$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle = b \right\},$$
$$\mathsf{NO} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle \neq b \right\},$$

where $b \in \mathbb{F}$, for any $i \in [t]$, Λ_i is a description of a uniform circuit that on input $w \in \mathbb{F}^k$ outputs $\langle \lambda_i, w \rangle$, and we view $(\Lambda_1, \ldots, \Lambda_t)$ as the explicit input, and $(b, C^{\otimes t}(x))$ as the implicit input.

Assuming that C can be encoded in time T, and each Λ_i is a $\mathrm{poly}(T')$-uniform circuit of size T', the prover has size $O(t \cdot n^{t-1} \cdot (T + T')) + \mathrm{poly}(T, T', t)$, and the verifier has size $\mathrm{poly}(\log(T), \log(T'), t/\delta)$.

[23] We emphasize that this includes the degree 1 extension, that is, when $\mathbb{F}' = \mathbb{F}$.

Lemma 5.2 follows by composing the sumcheck protocol for rank 1 tensor coefficients from [RR20] with an inner PCPP to reduce the verifier running time and query complexity. For completeness, we give a full proof of Lemma 5.2 in Appendix C. We proceed to the proof of Lemma 5.1, based on Lemma 5.2.

Proof of Lemma 5.1. The protocol $(\mathcal{P}, \mathcal{V})$ is given in Fig. 1. The protocol proceeds as follows. The verifier first picks a random codeword entry $\bar{i} \in [n]^t$ of $C^{\otimes t}$, and sends it to the prover. The parties then engage in a protocol to verify that $c := C^{\otimes t}(x)$ and $c'' := C^{\otimes t}(x')$ agree on the $\bar{i}$ coordinate. If $x = x'$, then we clearly have that $c'' = C^{\otimes t}(x') = C^{\otimes t}(x) = c$, and so c'' and c agree on all their coordinates. On the other hand, if $x \neq x'$ then since $C^{\otimes t}$ has relative distance δ^t, $c'' = C^{\otimes t}(x')$ and $c = C^{\otimes t}(x)$ must differ on at least a δ^t-fraction of the coordinates. Consequently, the verifier will pick a coordinate $\bar{i}$ on which c'' and c differ with probability at least δ^t.

To verify that $c''(\bar{i}) = c(\bar{i})$ we make use of the sumcheck protocol for rank 1 tensor coefficients given in Lemma 5.2. The main observation is that by the structure of tensor codes, there are coefficients $\lambda_1, \ldots, \lambda_t \in \mathbb{F}^k \subseteq (\mathbb{F}')^k$ so that the $\bar{i}$ coordinate of $c'' = C^{\otimes t}(x')$ can be expressed as $\langle \lambda_1 \otimes \cdots \otimes \lambda_t, x' \rangle$. Consequently, verifying that $c''(\bar{i}) = c(\bar{i})$ can be done by executing the sumcheck protocol from Lemma 5.2 on the codeword $c' := (C')^{\otimes t}(x')$ with coefficients $\lambda_1, \ldots, \lambda_t$ and $b = c(\bar{i})$.

ConsistencyCheck

- ***Prover $\mathcal{P}$'s Input:*** A pair of codewords $c := C^{\otimes t}(x)$ and $c' := (C')^{\otimes t}(x')$
- ***Verifier $\mathcal{V}$'s Input:*** Oracle access to c, c'

1. The verifier sends a uniform random index $\bar{i} \in [n]^t$.
2. For $j \in [t]$, let $\lambda_j \in \mathbb{F}^k \subseteq (\mathbb{F}')^k$ be such that $(C(w))_{i_j} = \langle \lambda_j, w \rangle$, and note that by our assumptions, there exists a poly(T)-uniform circuit Λ_j of size T that on input w outputs $\langle \lambda_j, w \rangle$.
3. $\mathcal{P}$ and $\mathcal{V}$ execute the protocol Π given in Lemma 5.2 with the code C', codeword c', $b = c(\bar{i})$, and $\Lambda_1, \ldots, \Lambda_t$. The verifier $\mathcal{V}$ accepts if and only if the verifier in Π accepts.

Figure 1: Consistency Checking

The round complexity, query complexity, and prover and verifier sizes are an immediate consequence of Lemma 5.2. Next we show completeness and soundness.

The main observation that enables us to utilize Lemma 5.2 is that by the properties of tensor codes, for any message $w \in \mathbb{F}^{k^t}$, the $\bar{i} = (i_1, \ldots, i_t)$ coordinate of $C^{\otimes t}(w)$ can be expressed as

$$(C^{\otimes t}(w))(\bar{i}) = \sum_{j_1, \ldots, j_t \in [k]} \lambda_1(j_1) \cdots \lambda_t(j_t) \cdot w(j_1, \ldots, j_t) = \langle \lambda_1 \otimes \cdots \otimes \lambda_t, w \rangle. \tag{3}$$

Completeness. Suppose that $x = x'$. In that case, by Eq. (3) for any choice of $\bar{i} \in [n]^t$ we have that

$$\langle \lambda_1 \otimes \cdots \otimes \lambda_t, x' \rangle = \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle = c(\bar{i}) = b,$$

and so, by the completeness condition of Lemma 5.2, the verifier of Π will accept with probability 1. Consequently, $\mathcal{V}$ will accept with probability 1 as well.

Soundness. Suppose that $x \neq x'$. Then since C has relative distance δ, we have that $C^{\otimes t}$ has relative distance δ^t, and so $c = C^{\otimes t}(x)$ and $c'' := C^{\otimes t}(x')$ differ on at least a δ^t-fraction of the coordinates. This implies in turn that with probability at least δ^t over the choice of $\bar{i}$, it holds that $c(\bar{i}) \neq c''(\bar{i})$. But if this is the case, then we have that

$$\langle \lambda_1 \otimes \cdots \otimes \lambda_t, x' \rangle = c''(\bar{i}) \neq c(\bar{i}) = b,$$

and so, by the soundness condition of Lemma 5.2, the verifier of Π will reject with probability at least $\frac{(\delta')^t}{2}$. Consequently, $\mathcal{V}$ will reject with probability at least $\frac{(\delta \cdot \delta')^t}{2}$. □

5.2 Multi-sumcheck protocol

In this section we prove Lemma 4.8 by constructing a multi-sumcheck protocol with constant overhead. Before we present the protocol, we describe the multiplication codes we use, and point out the differences from the high-level overview given in Section 1.3.

Recall that in the overview given in Section 1.3, we assumed that the codes $\{E_\ell\}_\ell$ are multiplication codes, in the sense that for each code $E = E_\ell$, the set $\{e \star e' : e, e' \in E\}$ spans a code $E^\star$ of sufficiently high distance, where the star symbol refers to *pointwise multiplication*. While in principle we could have used such codes for our protocol (which would also simplify the presentation), all codes we are aware of satisfying this property exactly (e.g., polynomial-based codes such as Reed-Solomon, Reed-Muller, or algebraic geometry codes) are defined over larger alphabets, while we would like the initial codes E_ℓ in the sequence to have small alphabet — in particular, we would like for E_0 to be a binary code.

To this end, we implicitly use in our protocol the tensor-based construction of Meir [Mei13] that works over any finite field, including the binary field. In more detail, recall that in our code construction (cf., Section 4.1) we did not require the codes E_ℓ to be multiplication codes. To turn these codes into (a variant of) multiplication codes we replace the star operation $\star$ with the tensor-product operation $\otimes$, where for a pair of vectors $e, e' \in \mathbb{F}^n$, their tensor product $e \otimes e' \in \mathbb{F}^{n \times n}$ is given as $(e \otimes e')(i, j) = e_i \cdot e'_j$ for any $i, j \in [n]$. The first observation is that the span of $\{e \otimes e' : e, e' \in E\}$ is contained in the tensor code $E^{\otimes 2}$, and that $E^{\otimes 2}$ has high relative distance, provided that E has high relative distance. The second observation is that the pointwise product $e \star e'$ appears as a substring of $e \otimes e'$ (namely on the diagonal). It is not hard to see that these two properties suffice for executing the multi-sumcheck protocol described in Section 1.3.

In more detail, assume for simplicity that $d = 2$, and recall that in the multi-sumcheck protocol the prover is supposed to first send the codeword $w := \sum_i c(i, \cdot) \otimes c'(i, \cdot) \in E^{\otimes 2}$. In fact, it suffices for the prover to send the systematic part of w (from which the verifier can generate any entry of w), which is given in Eq. (4). The verifier then checks that the substring of w that corresponds to the pointwise multiplication of the systematic parts of c and c' sums up to the given value b. If this is the case, then the verifier chooses a random codeword entry (r, r') in $E^{\otimes 2}$, and computes $w(r, r')$ out of the systematic part of w.[24] Finally, the parties continue to the next iteration with the restriction of c, c' to the r, r' columns, respectively.

The multi-sumcheck protocol is given in Fig. 2 below.

We start by showing completeness and soundness for our protocol. To this end, we shall need the following technical claim.

[24] Here we use that E is a Reed-Solomon code, and consequently each individual codeword symbol in E can be computed in time that only depends logarithmically on the field size.

MultiSumcheck

- ***Prover $\mathcal{P}$'s Input:*** A bit $b \in \{0,1\}$ and codewords $c_1, \ldots, c_d \in C_0$.
- ***Verifier $\mathcal{V}$'s Input:*** The same bit b and oracle access to the same codewords $c_1, \ldots, c_d$.

1. Set $b_0 \leftarrow b$ and $c_{0,j} \leftarrow c_j$, for every $j = 1, \ldots, d$.
2. For $\ell = 0, 1, ..., \log_a(n) - 2$:
   (a) Consider the d-dimensional cube $M_\ell : [a]^d \to \mathbb{F}_\ell$ defined as:
   $$M_\ell(j_1, j_2, \ldots, j_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, j_1) \cdots c_{\ell,d}(i, j_d) \tag{4}$$
   for every $j_1, \ldots, j_d \in [a]$, where each $c_{\ell,j} \in C_\ell$ is viewed as a codeword in the two-dimensional tensor product $D_\ell \otimes E_\ell$. The prover computes M_ℓ and sends it to the verifier.
   (b) The verifier receives the cube $\tilde{M}_\ell$, which is allegedly equal to M_ℓ. It checks that $\sum_{j \in [a]} \tilde{M}_\ell(j, \ldots, j) = b_\ell$. If not, then it rejects and aborts.
   (c) The verifier chooses a random codeword entry $(r_1, \ldots, r_d)$ in $E_\ell^{\otimes d}$ and sends it to the prover, and both parties set $b_{\ell+1}$ to be the value of the $(r_1, \ldots, r_d)$ coordinate of $E_\ell^{\otimes d}(\tilde{M}_\ell)$.
   (d) For $j = 1, \ldots, d$:
      i. Let $\hat{c}_{\ell,j}$ denote the restriction of $c_{\ell,j}$ to the first $\frac{n}{a^{\ell+1}}$ coordinates in the r_j'th column (i.e., the systematic part of the r_j'th column). The prover computes $c_{\ell+1,j} = C_{\ell+1}(\hat{c}_{\ell,j})$, and sends the resulting codeword to the verifier.
      ii. The parties engage in the Consistency Checking Protocol of Lemma 5.1 on inputs $c_{\ell,j}(\cdot, r_j) \in D_\ell$ and $c_{\ell+1,j} \in C_{\ell+1}$. If the verifier in the Consistency Checking Protocol rejects, then the verifier rejects and aborts.
3. For $\ell = \log_a(n) - 1$, the verifier explicitly checks that $\sum_{i \in [a]} c_{\ell,1}(i) \cdots c_{\ell,d}(i) = b_\ell$. If not, it rejects; Otherwise, it accepts.

Figure 2: Multi-sumcheck Protocol

Claim 5.3. *For any $\ell = 1, \ldots, \log_a(n) - 2$, and for any codeword entry $(r_1, \ldots, r_d)$ of $E_\ell^{\otimes d}$, we have that*

$$\left(E_\ell^{\otimes d}(M_\ell)\right)(r_1, \ldots, r_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d).$$

Proof. In what follows, let a' denote the codeword length of E_ℓ, let $e = E_\ell^{\otimes d}(M_\ell) \in \mathbb{F}^{(a')^d}$, and let $e' \in \mathbb{F}^{(a')^d}$ be defined as

$$e'(r_1, \ldots, r_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d)$$

for any $r_1, \ldots, r_d \in [a']$. Our goal is to show that $e = e'$.

We first claim that e' is a codeword of $E_\ell^{\otimes d}$. To see this, note that for any $j \in [d]$, the restriction of e' to any axis-parallel line in direction j is a linear combination of codewords of the form $c_{\ell,j}(i, \cdot) \in E_\ell$ for $i \in [n/a^{\ell+1}]$. By linearity of E_ℓ, this implies in turn that the restriction of e' to any axis-parallel line is a codeword of E_ℓ, and consequently, by Fact 2.15, e' is a codeword of $E_\ell^{\otimes d}$. So we conclude that both e and e' are codewords of $E_\ell^{\otimes d}$.

Next we argue that e and e' agree on their systematic parts, and consequently, since both are codewords of $E_\ell^{\otimes d}$, we must have that $e = e'$. To see this, note that by the definition of M_ℓ given in Eq. (4), and recalling that $E_\ell^{\otimes d} : \mathbb{F}^{a^d} \to \mathbb{F}^{(a')^d}$ is a systematic code, we have that

$$e(r_1, \ldots, r_d) = M_\ell(r_1, \ldots, r_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d) = e'(r_1, \ldots, r_d)$$

for any $r_1, \ldots, r_d \in [a]$. We conclude that the codewords e and e' agree on their systematic part of size $[a]^d$, and consequently $e = e'$, which concludes the proof of the claim. □

Completeness. Completeness relies on the following claim.

Claim 5.4. *Suppose that $\sum_{i \in [n]} c_1(i) \cdots c_d(i) = b$. Then when $\mathcal{V}$ interacts with $\mathcal{P}$, for any $\ell = 0, 1, \ldots, \log_a(n) - 1$, it holds that:*

$$\sum_{i \in [n/a^\ell]} c_{\ell,1}(i) \cdots c_{\ell,d}(i) = b_\ell. \tag{5}$$

Proof. We prove by induction on ℓ. For $\ell = 0$, Eq. (5) just follows by our assumption that $\sum_{i \in [n]} c_1(i) \cdots c_d(i) = b$, and our initial setting of $b_0 \leftarrow b$ and $c_{0,j} \leftarrow c_j$ for $j = 1, \ldots, d$.

Next assume that Eq. (5) holds for some $\ell \in \{0, 1, \ldots, \log_a(n) - 2\}$, and we shall show that it holds for $\ell + 1$ as well. To see this, first note that by Step 2c and Claim 5.3, we have that

$$b_{\ell+1} = \left(E_\ell^{\otimes d}(M_\ell)\right)(r_1, \ldots, r_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d).$$

On the other hand, by Step 2(d)i we have that $c_{\ell+1,j} = C_{\ell+1}(\hat{c}_{\ell,j})$ for all $j \in [d]$, and so we have that

$$\sum_{i \in [n/a^{\ell+1}]} c_{\ell+1,1}(i) \cdots c_{\ell+1,d}(i) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d).$$

Combining the above two equalities gives the desired conclusion that

$$\sum_{i\in[n/a^{\ell+1}]} c_{\ell+1,1}(i)\cdots c_{\ell+1,d}(i) = b_{\ell+1}.$$

□

Next assume that $\sum_{i\in[n]} c_1(i)\cdots c_d(i) = b$ (that is, the original statement is true), we will use the above Claim 5.4 to show that in this case the verifier will accept with probability 1.

First, by the definition of M_ℓ given in Eq. (4) and Claim 5.4, in Step 2b we have that

$$\sum_{j\in[a]} M_\ell(j,\ldots,j) = \sum_{j\in[a]}\sum_{i\in[n/a^{\ell+1}]} c_{\ell,1}(i,j)\cdots c_{\ell,d}(i,j) = \sum_{i\in[n/a^{\ell}]} c_{\ell,1}(i)\cdots c_{\ell,d}(i) = b_\ell,$$

and consequently the verifier will not reject in this step.

Moreover, by Step 2(d)i, we have that both codewords $c_{\ell,j}(\cdot, r_j) \in D_\ell$ and $c_{\ell+1,j} \in C_{\ell+1}$ are encodings of the same messages $\hat{c}_{\ell,j}$, and consequently, by the completeness of the Consistency Checking Protocol, the verifier will not reject in this step.

Finally, by Claim 5.4, we also have that $\sum_{i\in[n/a^\ell]} c_{\ell,1}(i)\cdots c_{\ell,d}(i) = b_\ell$ for $\ell = \log_a(n) - 1$, and so the verifier will accept with probability 1 in Step 3.

We conclude that in case $\sum_{i\in[n]} c_1(i)\cdots c_d(i) = b$, the verifier accepts with probability 1 when interacting with the honest prover.

Soundness. We shall show that the protocol has soundness error at most $1-\gamma_0$ for some absolute constant $\gamma_0 > 0$. In what follows, fix a deterministic prover strategy $\mathcal{P}^*$. Soundness relies on the following claim.

Claim 5.5. *Suppose that the following holds in some round $\ell \in \{0,1,\ldots,\log_a(n)-2\}$:*

1. $\sum_{i\in[n/a^\ell]} c_{\ell,1}(i)\cdots c_{\ell,d}(i) \neq b_\ell$.
2. *In Step 2b, the verifier's check passes, that is, $\sum_{j\in[a]} \tilde{M}_\ell(j,\ldots,j) = b_\ell$.*
3. *In Step 2(d)i, the prover sends $c_{\ell+1,j} = C_{\ell+1}(\hat{c}_{\ell,j})$ for all $j \in [d]$.*

Then, with probability at least $(\delta_\ell)^d$ over the choice of $r_1,\ldots,r_d$ in Step 2c, it holds that

$$\sum_{i\in[n/a^{\ell+1}]} c_{\ell+1,1}(i)\cdots c_{\ell+1,d}(i) \neq b_{\ell+1},$$

where δ_ℓ denotes the relative distance of E_ℓ.

Proof. By the definition of M_ℓ given in Eq. (4), and our first assumption that $\sum_{i\in[n/a^\ell]} c_{\ell,1}(i)\cdots c_{\ell,d}(i) \neq b_\ell$, we have that

$$\sum_{j\in[a]} M_\ell(j,\ldots,j) = \sum_{j\in[a]}\sum_{i\in[n/a^{\ell+1}]} c_{\ell,1}(i,j)\cdots c_{\ell,d}(i,j) = \sum_{i\in[n/a^{\ell}]} c_{\ell,1}(i)\cdots c_{\ell,d}(i) \neq b_\ell.$$

On the other hand, by our second assumption that the verifier's check in Step 2b passes we have that $\sum_{j\in[a]} \tilde{M}_\ell(j,\ldots,j) = b_\ell$.

We conclude that $M_\ell \neq \tilde{M}_\ell$, and so $E_\ell^{\otimes d}(M_\ell)$ and $E_\ell^{\otimes d}(\tilde{M}_\ell)$ are two different codewords of $E_\ell^{\otimes d}$. Since by Fact 2.13, $E_\ell^{\otimes d}$ has relative distance at least $(\delta_\ell)^d$, this implies in turn that $E_\ell^{\otimes d}(M_\ell)$ and $E_\ell^{\otimes d}(\tilde{M}_\ell)$ differ on at least a $(\delta_\ell)^d$-fraction of coordinates. Consequently, we have that

$$b_{\ell+1} = \left(E_\ell^{\otimes d}(\tilde{M}_\ell)\right)(r_1, \ldots, r_d) \neq \left(E_\ell^{\otimes d}(M_\ell)\right)(r_1, \ldots, r_d) \tag{6}$$

with probability at least $(\delta_\ell)^d$ over the choice of $r_1, \ldots, r_d$.

Next assume that Eq. (6) holds. By Claim 5.3, we have that

$$\left(E_\ell^{\otimes d}(M_\ell)\right)(r_1, \ldots, r_d) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d).$$

On the other hand, by our third assumption, we have that $c_{\ell+1,j} = C_{\ell+1}(\hat{c}_{\ell,j})$ for all $j \in [d]$, and so we have that

$$\sum_{i \in [n/a^{\ell+1}]} c_{\ell+1,1}(i) \cdots c_{\ell+1,d}(i) = \sum_{i \in [n/a^{\ell+1}]} c_{\ell,1}(i, r_1) \cdots c_{\ell,d}(i, r_d).$$

Combining the above two equalities with Eq. (6) gives the desired conclusion that

$$\sum_{i \in [n/a^{\ell+1}]} c_{\ell+1,1}(i) \cdots c_{\ell+1,d}(i) = \left(E_\ell^{\otimes d}(M_\ell)\right)(r_1, \ldots, r_d) \neq b_{\ell+1}.$$

□

Next assume that $\sum_{i \in [n]} c_1(i) \cdots c_d(i) \neq b$ (that is, the original statement is false). We will use Claim 5.5 to show that in this case the verifier will reject with probability at least γ_0, for an absolute constant $\gamma_0 > 0$.

We first argue that we may assume that Conditions 2 and 3 in Claim 5.5 above are satisfied in any round $\ell \in \{0, 1, \ldots, \log_a(n) - 2\}$. To see this, first note that we may assume that in any round $\ell \in \{0, 1, \ldots, \log_a(n) - 2\}$, the verifier's check in Step 2b passes, since otherwise the verifier immediately rejects. Next we claim that we may further assume that the third condition holds.

Suppose that, on the contrary, in some round $\ell \in \{0, 1, \ldots, \log_a(n) - 2\}$, in Step 2(d)i, the prover sends for some $j \in [d]$ a codeword of $C_{\ell+1}$ other than $C_{\ell+1}(\hat{c}_{\ell,j})$. Then in this case, by the soundness guarantee of the Consistency Checking Protocol given in Lemma 5.1, the verifier in the Consistency Checking Protocol rejects with probability at least $\frac{1}{2} \cdot (\mathrm{dist}(D_\ell) \cdot \mathrm{dist}(C_{\ell+1}))$. Recalling that by Propositions 4.2 and 4.6, both D_ℓ and $C_{\ell+1}$ have relative distance at least $(\alpha_0)^t$ for an absolute constant $\alpha_0 > 0$, and that t is a constant, we conclude that the verifier rejects with constant probability. Hence we may also assume that in every round $\ell \in \{0, 1, \ldots, \log_a(n) - 2\}$, in Step 2(d)i, the prover sends $C_{\ell+1}(\hat{c}_{\ell,j})$ for all $j \in [d]$.

Next observe that by our assumption that $\sum_{i \in [n]} c_1(i) \cdots c_d(i) \neq b$, and our initial setting of $b_0 \leftarrow b$ and $c_{0,j} \leftarrow c_j$ for $j = 1, \ldots, d$, we have that $\sum_{i \in [n]} c_{0,1}(i) \cdots c_{0,d}(i) \neq b_0$. But given this, by Claim 5.5, we have that $\sum_{i \in [a]} c_{\ell,1}(i) \cdots c_{\ell,d}(i) \neq b_\ell$, for every $\ell \in \{0, 1, \ldots, \log_a(n) - 1\}$, with probability at least $\beta_0 := \prod_{\ell=0}^{\log_a(n)-2} (\delta_\ell)^d$. Consequently, the check in Step 3 will fail with probability at least β_0.

Next we show that β_0 is greater than some absolute constant. To see that this is the case, recall that by Proposition 4.4, $\delta_0 = (\alpha_0)^t$ for an absolute constant $\alpha_0 > 0$, and

$$\delta_\ell = \left(1 - \frac{1}{|\mathbb{F}_\ell|^{1/t}}\right)^t \geq \left(1 - q^{-\ell/t}\right)^t$$

for every $\ell \geq 1$. Consequently, we have that

$$\begin{aligned}
\beta_0 &\geq (\alpha_0)^{td} \cdot \prod_{\ell=1}^{\infty} \left(1 - q^{-\ell/t}\right)^{td} \\
&\geq (\alpha_0)^{td} \cdot \prod_{\ell=1}^{\infty} \left(1 - tdq^{-\ell/t}\right) \\
&\geq (\alpha_0)^{td} \cdot \left(1 - td \sum_{\ell=1}^{\infty} q^{-\ell/t}\right) \\
&= (\alpha_0)^{td} \cdot \left(1 - td \cdot \frac{q^{-1/t}}{1 - q^{-1/t}}\right),
\end{aligned}$$

where the last expression is lower bounded by $\frac{(\alpha_0)^{td}}{3}$ by our assumption that $q \geq (4td)^t$. Finally, we note that $\beta_0 = \Omega(1)$ for any choice of constants d and t.

Next we analyze the verifier and prover sizes.

Verifier size. For every iteration ℓ, computing the sum $\sum_{j \in [a]} \tilde{M}_\ell(j, \ldots, j)$ in Step 2b takes time $O(a^d \cdot \log(|\mathbb{F}_\ell|)) = \mathrm{poly}(q^{td}, \ell)$ by our choice of $a = q^{t-1}$ and $|\mathbb{F}_\ell| \leq q^{2\ell}$. By Proposition 4.4 and Claim 2.16, computing the value of the $(r_1, \ldots, r_d)$ coordinate in $E_\ell^{\otimes d}(\tilde{M}_\ell)$ takes time $a^d \cdot \mathrm{poly}(a, \log(|\mathbb{F}_\ell|)) = \mathrm{poly}(q^{td}, \ell)$. Lastly, by Lemma 5.1, Proposition 4.1, and Proposition 4.5, the verifier's running time in the Consistency Checking Protocol executed in Step 2(d)ii is $\mathrm{poly}(\log n, \log a, t, \log(|\mathbb{F}_\ell|)) = \mathrm{poly}(\log n, q, t, \ell)$. Overall, we get a running time that is upper bounded by $\sum_{\ell=0}^{\log n} \mathrm{poly}(\log n, q^{td}, \ell) = \mathrm{poly}(\log n, q^{td})$, which is at most $\mathrm{polylog}(n)$, since d, t, and q are constants. This also implies that the verifier size is at most $\mathrm{polylog}(n)$.

Prover size. For every iteration ℓ, computing the cube M_ℓ in Step 2a can be implemented by a Boolean circuit of size $a^d \cdot \frac{n}{a^{\ell+1}} \cdot \mathrm{polylog}(|\mathbb{F}_\ell|) = \frac{n}{q^{t\ell}} \cdot \mathrm{poly}(q^{td}, \ell)$ by our choice of $a = q^{t-1}$ and $|\mathbb{F}_\ell| \leq q^{2\ell}$. By Proposition 4.4 and Claim 2.16, computing the value of the $(r_1, \ldots, r_d)$ coordinate in $E_\ell^{\otimes d}(M_\ell)$ can be implemented by a Boolean circuit of size $a^d \cdot \mathrm{poly}(a, \log(|\mathbb{F}_\ell|)) = \mathrm{poly}(q^{td}, \ell)$. By Proposition 4.6, encoding the new codeword in Step 2(d)i can be implemented by a Boolean circuit of size $\frac{n}{a^{\ell+1}} \cdot \mathrm{poly}(a, 2^t, |\mathbb{F}_{\ell+1}|) = \frac{n}{q^{t\ell}} \cdot \mathrm{poly}(q^t, q^\ell)$. Lastly, by Lemma 5.1, Proposition 4.1, and Proposition 4.5, the prover in the Consistency Checking Protocol executed in Step 2(d)ii can be implemented by a Boolean circuit of size

$$\frac{n}{a^\ell} \cdot \mathrm{poly}(a, |\mathbb{F}_\ell|, 2^t) + \mathrm{poly}\left(\left(\frac{n}{a^\ell}\right)^{1/t}, a^{1/t}, |\mathbb{F}_\ell|^{1/t}, \log(|\mathbb{F}_\ell|), t\right) = \frac{n}{q^{t\ell}} \cdot \mathrm{poly}(q^t, q^\ell),$$

where the last equality holds for a sufficiently large constant t.

Overall, we get a circuit size of at most

$$\sum_{\ell=0}^{\log n} \frac{n}{q^{t\ell}} \cdot \mathrm{poly}(q^{td}, q^{\ell}) \leq n \cdot q^{O(td)} \sum_{\ell=0}^{\log n} \frac{q^{O(\ell)}}{q^{t\ell}} \leq n \cdot q^{O(td)} \sum_{\ell=0}^{\infty} q^{-(t-O(1))\cdot\ell} = O(n),$$

where the last equality holds for any sufficiently large constant t.

Finally, note that the round complexity is clearly $O(\log n)$ (recalling that the round complexity of the consistency checking protocol is constant). As to the query complexity, note that the verifier makes a constant number of queries to the input codewords of C_0 in Step 2(d)ii in the first round as part of the consistency checking protocol. Similarly, the verifier makes a constant number of queries to each codeword of $\mathcal{C}$, sent by the prover, in Step 2(d)ii as part of the consistency checking protocol. The rest of the prover messages are either the messages M_ℓ sent in Step 2a which the verifier reads in their entirety, or the messages sent in Step 2(d)ii as part of the consistency checking protocol to which the verifier makes a constant number of queries.

6 Fast IOP for circuit satisfiability

In this section we prove Theorem 3.1 by constructing an IOP for circuit satisfiability with a linear-size prover. We start by re-stating the theorem.

Theorem 3.1. *Let $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ be a size $S(n,m) \geq n+m$ Boolean circuit. There exists an $O(\log S)$-round IOP for the language $\left\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ with constant soundness error. The prover, given as auxiliary input also the witness w, can be implemented by a size $O(S)$ Boolean circuit. After a size $O(S)$ private pre-processing step, the verifier can be implemented by a size $O(n) + \mathrm{polylog}(S)$ Boolean circuit, and has $\mathrm{polylog}(S)$ query complexity.*

Section Organization. We start in Section 6.1 with an overview of the arithmetization technique, and how to reduce circuit satisfiability to multi-sumcheck. This overview is optional and can be skipped. The subsequent sections are devoted to the formal proof of Theorem 3.1. In Section 6.2 we develop a pair of sub-protocols that will be useful for the proof. In Section 6.3 we use these tools to obtain an IOPP for *circuit evaluation* and use the latter in Section 6.4 to prove Theorem 3.1.

6.1 From multi-sumcheck to circuit satisfiability: an overview

In this subsection we provide an overview of our arithmetization step, which is loosely based on [BCG+17a]. We remark that the overview may be read independently of the other technical sections.

Let C be a circuit of size S. For simplicity and without loss of generality we assume that C consists only of NAND gates.[25] We show how to construct an IOP proving to the verifier, who holds x, that there exists w such that $\mathsf{C}(x,w) = 1$.

Given an input x and witness w, let $W \in \{0,1\}^S$ be the values obtained by all of the gates of C when evaluated on input (x,w) (where we assume some fixed ordering of the gates of C). Let W_R be the same as W but shifted based on the circuit topology of C as follows: for every $i \in [S]$, set $W_R(i) = W(\pi_R(i))$, where $\pi_R(i)$ is the index of the gate that serves as the right input to gate i. We

[25] Recall that any constant-size gate can be emulated by a constant size sub-circuit that uses only NAND gates.

define W_L similarly, but wrt the left input gate. For ease of notation and simplicity, we assume for this overview that π_R and π_L are permutations on $[S]$ (although in the actual construction, given that the fan-out is greater than 1, they may not be injective).

The construction utilizes a linear-time encodable code $\mathcal{C}$ supporting a linear-size multi-sumcheck. Constructing such a code is our main contribution and an overview of the construction is provided in Section 1.3.1.

In our IOP for circuit satisfiability, the prover first computes and sends $\widehat{W} = \mathcal{C}(W)$, $\widehat{W}_R = \mathcal{C}(W_R)$ and $\widehat{W}_L = \mathcal{C}(W_L)$. Since $\mathcal{C}$ is linear-time encodable, and the permutations are fixed based on the circuit topology, this step can be implemented in linear-time.[26] The verifier now needs to check that:

1. $\widehat{W}$, $\widehat{W}_R$ and $\widehat{W}_L$ are all valid codewords.
2. $\widehat{W}_R$ is consistent with $\widehat{W}$.
3. $\widehat{W}_L$ is consistent with $\widehat{W}$.
4. $\widehat{W}(i) = \mathsf{NAND}\big(\widehat{W}_L(i), \widehat{W}_R(i)\big) = 1 - \widehat{W}_L(i) \cdot \widehat{W}_R(i)$, for every $i \in [S]$ (other than the input gates, but let us ignore those gates here).
5. The inputs of $\widehat{W}$ are correct (i.e., are equal to x) and the output value is 1.

Test 1 can be handled using standard local testing and (relaxed) local correction techniques. In slightly more detail, we can ensure that $\mathcal{C}$ is a tensor code and employ results of [Vid15, GRR18] (see also [RR20]).

To handle Test 2 (Test 3 can be handled similarly), the verifier chooses at random $r \in \{0,1\}^S$ and sends r to the prover.[27] To check that $\widehat{W}_R$ and $\widehat{W}$ are consistent, it now suffices to check that

$$\sum_{i \in [S]} r(i) \cdot \widehat{W}(\pi_R(i)) = \sum_{i \in [S]} r(i) \cdot \widehat{W}_R(i)$$

(the test can be repeated a constant number of times to reduce the soundness error). To perform this test, the prover sends the value α that is allegedly equal to both sides of the equation. The parties then engage in the multi-sumcheck protocol twice. First, to check that $\sum_{i \in [S]} r(i) \cdot \widehat{W}_R(i) = \alpha$, and then to check that $\sum_{i \in [S]} r(i) \cdot \widehat{W}(\pi_R(i)) = \alpha$. To see that the latter is indeed an instance of multi-sumcheck, we can rewrite it as $\sum_{i \in [S]} r'(i) \cdot \widehat{W}(i)$, where $r'(i) = r(\pi_R^{-1}(i))$. Note that for this step the verifier needs oracle access to $C(r)$ and $C(r')$ but these can be generated by the verifier in a preprocessing step (in linear time). This does however come at a cost - the entire preprocessing step must be kept hidden from the prover (until after the prover sends its first message), see also Remark 6.1.

For Test 4, the verifier similarly chooses a random $r \in \{0,1\}^S$ and the problem is reduced to checking that

$$\sum_{i \in [S]} r(i) \cdot \widehat{W}(i) = \sum_{i \in [S]} r(i) \cdot (1 - \widehat{W}_L(i) \cdot \widehat{W}_R(i)).$$

[26] Note that this means that the circuit C is effectively hardwired into the prover circuit, rather than being given as an input. This seems inherent since merely describing the circuit takes $O(S \cdot \log S)$ bits.

[27] In the actual protocol we choose r as the output of a small-bias generator (that can be generated in linear-time in its output) to save on the randomness complexity.

To perform this check, the parties employ the multi-sumcheck protocol to compute: (1) $\alpha = \sum_{i\in[S]} r(i)\cdot\widehat{W}(i)$, (2) $\beta = \sum_{i\in[S]} r(i)$ and (3) $\gamma = \sum_{i\in[S]} r(i)\cdot\widehat{W}_L(i)\cdot\widehat{W}_R(i)$. The verifier checks that $\alpha = \beta - \gamma$.[28]

There are several ways to handle Test 5. One relatively easy way is to have the verifier first encode the input x using a linear-time encodable code E, and then to apply the IOP wrt the language $\{E(x) : \exists w, \mathsf{C}(x,w) = 1\}$ (i.e., consisting of encodings of all satisfiable instances). Now, since the input is encoded, it suffices for the verifier to "spot check" that a constant number of the encoded input bits agree with the corresponding bits of W. Similarly, the output bit of W can be directly checked to be equal to 1.

Remark 6.1. *As mentioned above, the reason that the verifier requires a* private *pre-processing step is because it needs to generate $C(r)$ and $C(r')$, while keeping their values hidden from the prover. Indeed, it would be interesting to try to obtain a similar result with a* public *pre-processing step.*

6.2 Useful sub-protocols

In Section 6.2.1 we give a $\mathcal{C}$-encoded IOP for checking that two codewords of the code C_0 encode shifts of the same message, and (2) in Section 6.2.2 we give a $\mathcal{C}$-encoded IOP for verifying any pointwise relation between a constant number of codewords.

6.2.1 Shifted consistency check

The following lemma lets us check that messages encoded within two codewords of C_0 are shifts of one another.

Lemma 6.2. *Let $\pi : [n] \to [n]$ be a fixed function and consider the promise problem $(\mathsf{YES}_\pi, \mathsf{NO}_\pi)$, where*

$$\mathsf{YES}_\pi = \left\{ \big(C_0(x), C_0(y)\big) \ : \ \forall i \in [n],\ x(i) = y(\pi(i)) \right\}$$
$$\mathsf{NO}_\pi = (C_0 \times C_0) \backslash \mathsf{YES}_\pi.$$

There exists an $O(\log n)$-round $\mathcal{C}$-encoded IOP for $(\mathsf{YES}_\pi, \mathsf{NO}_\pi)$ with constant soundness error. The prover can be implemented by a size $O(n)$ Boolean circuit. After a size $O(n)$ private pre-processing step, the verifier can be implemented by a size polylog(n) *Boolean circuit.*

Furthermore, the verifier makes a constant number of queries to the input and to the pre-processed data, and for each prover message, the verifier either reads the entire message, or makes a constant number of queries to this message.

We emphasize that, in contrast to the description in Section 6.1, the function π can be arbitrary, and in particular does not need to be a permutation.

Proof of Lemma 6.2. Let $G : \{0,1\}^d \to \{0,1\}^n$ be the 0.49-evading set generator from Proposition 2.22 for $d = O(\log n)$. The $\mathcal{C}$-encoded IOP is described in Fig. 3.

[28] Actually, the value β can be computed off-line during the pre-processing step.

Shifted Consistency Check

Prover $\mathcal{P}$'s Input: A pair of codewords $C_0(x), C_0(y)$.

Verifier $\mathcal{V}$'s Input: Oracle access to $C_0(x), C_0(y)$.

Preprocessing Phase: The verifier chooses at random $s \in \{0,1\}^d$ and explicitly generates $r = G(s)$. The verifier also computes $r' \in \{0,1\}^n$, where $r'(i) = \sum_{j \in \pi^{-1}(i)} r(j)$ for every $i \in [n]$. The verifier generates the encodings $C_0(r)$ and $C_0(r')$. The preprocessed state is $(s, C_0(r), C_o(r'))$.

Online Phase:

1. The verifier sends s to the prover. The prover generates $C_0(r)$ and $C_0(r')$ similarly to the verifier.
2. The prover sends the value $\alpha = \sum_{i \in [n]} r(i) \cdot x(i)$ to the verifier.
3. The parties engage in the multi-sumcheck protocol of Lemma 4.8 to check that $\sum_{i \in [n]} r(i) \cdot x(i) = \alpha$ (with the verifier using oracle access to $C_0(x)$ from the input and to $C_0(r)$ from its pre-processed state).
4. The parties engage again in the multi-sumcheck protocol of Lemma 4.8, this time to check that $\sum_i r'(i) \cdot y(i) = \alpha$ (with the verifier using oracle access to $C_0(y)$ from the input and to $C_0(r')$ from its pre-processed state).
5. If the verifier rejected in any of the subroutines then reject, otherwise accept.

Figure 3: Shifted Consistency Checking

Before analyzing the protocol, let us first note that for every choice of $r \in \{0,1\}^n$ it holds that:

$$\sum_{i \in [n]} r'(i) \cdot y(i) = \sum_{i \in [n]} y(i) \sum_{j \in \pi^{-1}(i)} r(j) = \sum_{i \in [n]} r(i) \cdot y(\pi(i)), \tag{7}$$

where r' is defined as in the protocol (i.e., $r'(i) = \sum_{j \in \pi^{-1}(i)} r(j)$, for every $i \in [n]$).

Completeness. Assume that $x(i) = y(\pi(i))$ for every $i \in [n]$. Then, for every choice of r it holds that

$$\sum_{i \in [n]} r(i) \cdot x(i) = \sum_{i \in [n]} r(i) \cdot y(\pi(i)) = \sum_{i \in [n]} r'(i) \cdot y(i),$$

where the last equality is by Eq. (7). Thus, using the completeness of the multisumcheck protocol of Lemma 4.8, the verifier accepts in both Steps 3 and 4.

Soundness. Assume that there exists $i \in [n]$ such that $x(i) \neq y(\pi(i))$. Then, with probability at least 0.1 over the choice of the seed s, it holds that:

$$\sum_{i \in [n]} r(i) \cdot x(i) \neq \sum_{i \in [n]} r(i) \cdot y(\pi(i)) = \sum_{i \in [n]} r'(i) \cdot y(i),$$

where the equality is again due to Eq. (7). Assume that such a seed s was indeed chosen. Then, one of the claims checked in either Step 3 or 4 must be false and by the soundness of the protocol of Lemma 4.8, the verifier will reject with constant probability.

Prover Size. The prover first explicitly generates the entire string r which, by Proposition 2.22, can be done in linear size. The prover also computes r' which can be computed from r using the fixed function π in size $O(n)$. The prover also generates the encodings $C_0(r)$ and $C_0(r')$, which by Proposition 4.6 can be done in linear size. Steps 3 and 4 can likewise be implemented in linear size by Lemma 4.8.

Verifier Size. In its private pre-processing step, the verifier generates $C_0(r)$ and $C_0(r')$ which, as noted above, can be done by a linear-size circuit.

In the online phase, by Lemma 4.8, the verifier can be implemented by a polylog(n) size circuit. The furthermore clause, describing the verifier's query pattern follows directly from Lemma 4.8. □

6.2.2 Arithmetic consistency check

Next, we construct a $\mathcal{C}$-encoded IOP for verifying any pointwise relation between a constant number of codewords.

Lemma 6.3. *Let $d \in \mathbb{N}$ be a constant, $f : \{0,1\}^d \to \{0,1\}$ a function. Consider the promise problem $(\mathsf{YES}_f, \mathsf{NO}_f)$, where*

$$\mathsf{YES}_f = \Big\{ \big(C_0(x_1), \ldots, C_0(x_d)\big) \ : \ \forall i \in [n],\ f\big(x_1(i), \ldots, x_d(i)\big) = 0 \Big\}$$
$$\mathsf{NO}_f = (C_0 \times \cdots \times C_0) \backslash \mathsf{YES}.$$

There exists an $O(\log n)$-round $\mathcal{C}$-encoded IOP for $(\mathsf{YES}_f, \mathsf{NO}_f)$ with constant soundness error. The prover can be implemented by a size $O(n)$ Boolean circuit. After a size $O(n)$ private pre-processing step, the verifier can be implemented by a size polylog(n) *Boolean circuit.*

Furthermore, the verifier makes a constant number of queries to the input and to the pre-processed data, and for each prover message, the verifier either reads the entire message, or makes a constant number of queries to this message.

Proof. We first express f as a multilinear polynomial over the binary field. That is, $f(z) = \sum_{S \subseteq [d]} f_S \cdot z^S$, where $z^S = \prod_{j \in S} z(j)$, and $f_S \in \{0,1\}$ is the coefficient corresponding to the monomial z^S.

Let G be the 0.49-evading set generator from Proposition 2.22 with logarithmic seed length. The protocol proceeds as follows. The verifier chooses at random a seed s for G and sends it to the prover. Let $r = G(s)$. By Proposition 2.22 it suffices for the two parties to check that:

$$0 = \sum_{i \in [n]} r(i) \cdot f\big(x_1(i), \ldots, x_d(i)\big) = \sum_{S \subseteq [d]} f_S \sum_{i \in [n]} r(i) \cdot \prod_{j \in S} x_j(i). \tag{8}$$

The prover sends the values of $\alpha_S = \sum_{i \in [n]} r(i) \prod_{j \in S} x_j(i)$ for every $S \subseteq [d]$ and the verifier checks that indeed $\sum_{S \subseteq [d]} f_S \cdot \alpha_S = 0$.

To verify that each α_S was computed correctly, the two parties then run the multisumcheck protocol of Lemma 4.8, repeated in parallel $O(d)$ times to reduce the soundness error to $0.01 \cdot 2^{-d}$ (where the verifier uses its oracle access to $C_0(x_1), \ldots, C_0(x_d)$ and $C_0(r)$).

Completeness, soundness and the desired complexity follow in a straightforward way (similar to the proof of Lemma 6.2) from Lemma 4.8 and Proposition 2.22. □

6.3 IOPP for circuit evaluation

As our next step in proving Theorem 3.1, we construct a $\mathcal{C}$-encoded IOPP (an IOP of proximity, see Section 2.2) for *circuit evaluation*. This is captured by the following lemma.

Lemma 6.4. *Let $\mathsf{C} : \{0,1\}^n \to \{0,1\}$ be a size $S = S(n) \geq n$ Boolean circuit and $\delta > 0$ be a proximity parameter. There exists an $O(\log(S))$-round $\mathcal{C}$-encoded IOPP for the set $\left\{x \in \{0,1\}^n : \mathsf{C}(x) = 1\right\}$ with constant soundness error. The prover can be implemented by a size $O(S)$ Boolean circuit. After a size $O(S)$ private pre-processing step, the verifier can be implemented by a size $\mathrm{poly}(\log(S), 1/\delta)$ Boolean circuit.*

Furthermore, the verifier makes $O(1/\delta)$ queries to the input, a constant number of queries to the pre-processed data, and for each prover message, the verifier either reads the entire message, or makes a constant number of queries to this message.

Theorem 3.1 follows from Lemma 6.4 using techniques similar to the construction of PCPs for NP from PCPPs for $\mathcal{P}$ (see [BGH+06, DR06]), which are described in Section 6.4. We proceed to the proof of Lemma 6.4.

Let $\mathsf{C} : \{0,1\}^n \to \{0,1\}$ be a size S Boolean circuit. We assume for simplicity and without loss of generality that the internal gates of C are NAND gates with fan-in 2 and fan-out 2 (see Section 2.1 for additional details). Fix an ordering of the gates of C. We assume that the input gates are indexed by $1, \ldots, n$ and the output gate is indexed by S. In what follows when we refer to a gate $g \in [S]$ we simply mean the gate that is indexed by g in our ordering.

Let $\pi_L, \pi_R : [S] \to [S]$ be functions such that for every non-input gate g it holds that $\pi_L(g)$ (resp., $\pi_R(g)$) is the gate that corresponds to the left (resp., right) input of g. For an input gate g we define $\pi_L(g)$ and $\pi_R(g)$ arbitrarily.

Given an input $x \in \{0,1\}^n$ consider an evaluation of C on input x and let $W : [S] \to \{0,1\}$ correspond to the assignments given to each gate of the circuit in this evaluation. Let $W_R, W_L : [S] \to \{0,1\}$ be the assignment of the gates shifted according to the mappings π_L and π_R, respectively. That is, $W_R = W \circ \pi_R$ and $W_L = W \circ \pi_L$ (where $\circ$ refers to standard function composition). We say that a function $W : [S] \to \{0,1\}$ represents a *consistent computation* with respect to C if assigning the value $W(g)$ to every gate g is a consistent assignment: i.e., for every non-input gate g it holds that $W(g) = \mathsf{NAND}(W_L(g), W_R(g))$.

Define $\mathbf{1_n} = 1^n || 0^{S-n}$ (i.e., a sequence of n 1's followed by $S - n$ 0's). We define a function $f_{\mathsf{control}}(a, b, c, d)$ as follows:

1. If either $\big(d \leq n\big)$ or $\big(a = \mathsf{NAND}(b, c)\big)$ output 0.
2. Otherwise, output 1.

To gain some intuition on the definition of f_{control}, it is instructive to consider the following sequence of evaluations of f_{control}:

$$\Big(f_{\mathsf{control}}\big(W(i), W_L(i), W_R(i), \mathbf{1_n}(i)\big)\Big)_{i \in [S]}.$$

The definition of f_{control} ensures that if W is a consistent computation then this sequence of evaluations will all be 0, whereas if W represents an *inconsistent* computation then at least one will be 1. Indeed, the following fact is immediate from the definition of f_{control}.

Fact 6.5. *Let $W : [S] \to \{0,1\}$ and define $W_L = W \circ \pi_L$ and $W_R = W \circ \pi_R$. Then, W is a consistent computation if and only if*

$$f_{\mathsf{control}}\big(W(i), W_L(i), W_R(i), \mathbf{1_n}(i)\big) = 0$$

for every $i \in S$.

Using the above definitions and notations, we are now ready to describe the $\mathcal{C}$-encoded IOPP:

1. The prover evaluates the circuit to obtain the wire evaluations $W : [S] \to \{0,1\}$.
2. The prover computes $C_0(W)$, $C_0(W_L)$ and $C_0(W_R)$ and sends them (as oracles) to the verifier.
3. **Shifted consistency check:** The prover and verifier engage in the protocol of Lemma 6.2 twice, once to check that $W(\pi_L(\cdot)) \equiv W_L(\cdot)$ and once to check that $W(\pi_R(\cdot)) \equiv W_R(\cdot)$, where the verifier uses its oracle access to $C_0(W)$, $C_0(W_R)$ and $C_0(W_L)$ that were sent by the prover beforehand.
4. **Arithmetic consistency check** The prover and verifier engage in the protocol of Lemma 6.3 to check that:
$$f_{\mathsf{control}}\big(W(i), W_L(i), W_R(i), \mathbf{1_n}(i)\big) = 0$$
for every $i \in [S]$, where the verifier again uses its oracle access to $C_0(W)$, $C_0(W_R)$ and $C_0(W_L)$ that were sent by the prover beforehand and computes oracle queries to $C_0(\mathbf{1_n})$ by itself.
5. **Input proximity check:** Repeat $O(1/\delta)$ times: The verifier chooses at random $i \in [n]$ and checks that $W(i) = x(i)$ (using a single query to the input and single query to the systematic part of the codeword $C_0(W)$ that the prover sent).
6. **Output check:** The verifier checks that $W(S) = 1$ (using a single query to the systematic part of the codeword $C_0(W)$ that the prover sent).
7. If all of the verifiers checks pass then it accepts, and otherwise it rejects.

Completeness. Suppose that $\mathsf{C}(x) = 1$ and $C_0(W)$, $C_0(W_L)$ and $C_0(W_R)$ are as specified in the protocol. By construction it is indeed the case that $W(\pi_L(\cdot)) \equiv W_L(\cdot)$ and $W(\pi_R(\cdot)) \equiv W_R(\cdot)$ and so, by the completeness condition of Lemma 6.2, the verifier's checks in Step 3 pass. Similarly, by Fact 6.5 it holds that

$$f_{\mathsf{control}}(W(i), W_L(i), W_R(i), \mathbf{1_n}(i)) = 0$$

for every $i \in [S]$, and so by the completeness condition of Lemma 6.3 the verifier's checks in Step 4 pass. Since the first n bits of $C_0(W)$ are equal to the input x, and the S'th bit of $C_0(W)$ is equal to $\mathsf{C}(x) = 1$, the verifier's checks in Steps 5 and 6 also pass.

Soundness. Suppose that $x \in \{0,1\}^n$ is δ-far from the set $\{x' \in \{0,1\}^n : \mathsf{C}(x') = 1\}$ and fix a $\mathcal{C}$-encoded cheating prover strategy $\mathcal{P}^*$. Wlog we assume that $\mathcal{P}^*$ is deterministic and denote the first message that it sends by $(\hat{W}, \hat{Y}, \hat{Z})$. Note that since $\mathcal{P}^*$ is $\mathcal{C}$ encoded it holds that $\hat{W}, \hat{Y}, \hat{Z} \in C_0$. Further, denote their systematic parts (i.e., the messages encoded therein) by W, Y and Z.

Suppose first that either $W(\pi_L(\cdot)) \not\equiv Y(\cdot)$ or $W(\pi_R(\cdot)) \not\equiv Z(\cdot)$. Then, by the soundness condition of Lemma 6.2 the verifier rejects in Step 3 with constant probability. Thus, we may assume that $Y \equiv W_L$ and $Z \equiv W_R$.

Suppose that W represents an inconsistent computation. Then, by Fact 6.5 there exists some $i \in [S]$ for which

$$f_{\mathsf{control}}(W(i), Y(i), Z(i), \mathbf{1_n}(i)) \neq 0.$$

This means that the protocol of Lemma 6.3 is executed in Step 4 on a NO instance and so the verifier rejects with constant probability.

Thus, we may assume that W represents a consistent computation. Let x' be the first n bits of $\hat{W}$. Suppose that $\mathsf{C}(x') = 0$. Since W represents a consistent computation we have that $W(S) = 0$ and so the verifier rejects in Step 6. Thus, we may assume that $\mathsf{C}(x') = 1$, but this means that x' is δ-far from x and so the verifier rejects with constant probability in Step 5 as desired.

Prover Size. The prover first performs the computation which in particular generates all of the values of W. These values are then shifted according to both π_L and π_R (which are fixed functions) to produce W_L and W_R. This overall can be done by a size $O(S)$ circuit.

By Proposition 4.6 we can encode these messages using additional $O(S)$ size. The prover efficiency now follows from Lemmas 6.2 and 6.3. A circuit implementing the prover strategy is depicted in Fig. 4.

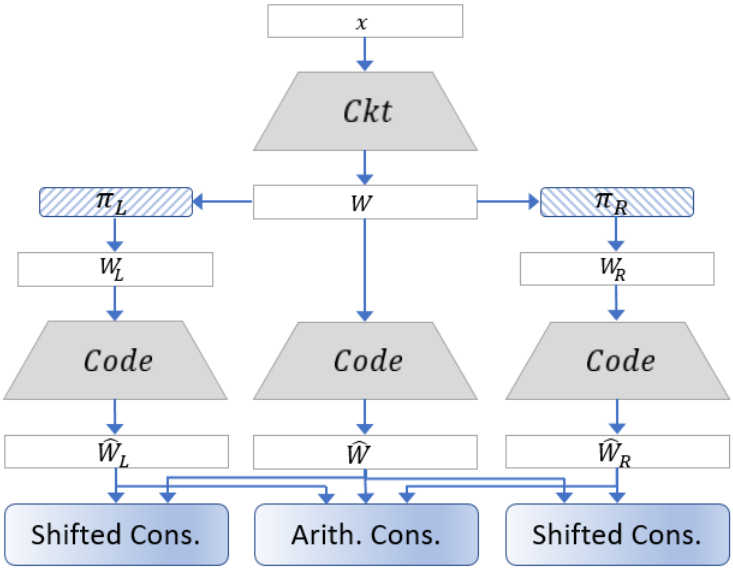

Figure 4: The Prover Circuit

Verifier Size. As part of its private pre-processing step, the verifier computes $C_0(\mathbf{1_n})$ and stores it as part of its preprocessed state. By Proposition 4.6, this can be done by an $O(S)$-size circuit. The rest of the pre-processing is just the pre-processing step of Lemmas 6.2 and 6.3, which can be done by an $O(S)$ size circuit.

In the online phase, the verifier in the protocol only executes the verifiers of Lemmas 6.2 and 6.3 as well as the input proximity check and output check. The former checks can be implemented by a polylog(S)-size, while the latter can be computed in time $O(\log(S)/\delta)$. The pattern of the verifier's queries follows from similar guarantees in Lemmas 6.2 and 6.3.

6.4 IOP for circuit satisfiability

We now turn to the proof of Theorem 3.1. Actually, since later on we will need some extra properties related to the query structure of the verifier, which do not appear in the statement of Theorem 3.1, we re-state the theorem here with an additional furthermore clause.

Lemma 6.6. *Let $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ be a size $S(n,m) \geq n+m$ Boolean circuit. There exists an $O(\log(S))$-round* IOP *for the language $\left\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ \mathsf{C}(x,w) = 1\right\}$ with constant soundness error. The prover, given as auxiliary input also the witness w, can be implemented by a size $O(S)$ Boolean circuit. After a size $O(S)$ private pre-processing step, the verifier can be implemented by a size $O(n) + \text{polylog}(S)$ Boolean circuit, and has $\text{polylog}(S)$ query complexity.*

Furthermore, the verifier makes a constant number of queries to the the pre-processed data, and for each prover message, the verifier either reads the entire message, or makes a constant number of queries to this message.

The rest of this subsection is devoted to the proof of Lemma 6.6, which immediately implies Theorem 3.1. We first construct a $\mathcal{C}$-encoded IOP for circuit satisfiability, and Lemma 6.6 then follows by applying Proposition 4.7.

Let $\mathsf{C} : \{0,1\}^{n+m} \to \{0,1\}$ be a circuit of size $S(n,m) \geq n+m$ and let E be a systematic binary linear-time encodable code from Theorem 2.20 with constant relative distance $\delta > 0$. Given an input $x \in \{0,1\}^n$, let $\hat{x} = E(x)$ and note that there is a linear-size circuit that checks that $\hat{x}$ is properly encoded (by re-encoding). Roughly speaking, our approach is for the prover to send the alleged witness w to the verifier and then for the two parties to run the IOPP of Lemma 6.4 on a circuit C' that gets as input $(\hat{x}, w)$, checks that $\hat{x}$ is properly encoded and that $C(x,w) = 1$. Intuitively, in case x is not "satisfiable", based on the distance of the code, $(\hat{x}, w)$ should be far from any accepting input.

A technical problem arises in case $\hat{x}$ is much shorter than w: in such a case the distance of E does not guarantee the distance of the pair $(\hat{x}, w)$ from an accepting pair. To get around this we actually make C' get as input sufficiently many copies of $\hat{x}$ to make their overall length roughly as long as w.

Let $\hat{n} = |\hat{x}|$ and let $\bar{x}$ consist of $\lceil m/\hat{n} \rceil$ copies of $\hat{x}$ and denote $\bar{n} = |\bar{x}| = \lceil m/\hat{n} \rceil \cdot \hat{n}$. We can bound $\bar{n}$ as follows.

Fact 6.7. $m \leq \bar{n} \leq \hat{n} + m$.

Proof. For the upper bound, observe that $\bar{n} = \lceil m/\hat{n} \rceil \cdot \hat{n} \leq (m/\hat{n} + 1) \cdot \hat{n} \leq m + \hat{n}$. For the lower bound, $\bar{n} \geq (m/\hat{n}) \cdot \hat{n} = m$. □

Consider a circuit $\mathsf{C}' : \{0,1\}^{\bar{n}+m} \to \{0,1\}$ which, given as input $(\bar{x}, w)$ checks that:

1. The input $\bar{x}$ consists of $\lceil m/\hat{n} \rceil$ copies of a string $\hat{x} \in \{0,1\}^{\hat{n}}$;
2. $\hat{x} \in E$; and

3. $\mathsf{C}(x, w) = 1$ where x is the message encoded by $\hat{x}$.

Note that C' can be implemented by a circuit of size $O(S(n+m))$ as follows: Step 1 can be directly implemented by a size $O(\hat{n}+m)$ circuit by simple equality checks. Step 2 can be implemented by a size $O(n)$ circuit by Theorem 2.20. Step 3 can be implemented by a size $S(n,m)$ circuit. Since $S(n,m) \geq n+m$ and $\hat{n} = O(n)$ we have that C' has size $O(S(n,m))$.

In our IOP, the prover sends the witness w and both parties (locally) compute $\hat{x}$. The parties then engage in the IOPP of Lemma 6.4 to check that $\mathsf{C}'(\bar{x}, w) = 1$ using proximity parameter $\delta/2$. The verifier and prover's queries to $\bar{x}$ are emulated by queries to $\hat{x}$ (by shifting indices accordingly).

The completeness as well as complexity analysis follow immediately from Lemma 6.4.[29] For soundness, suppose that x is a NO instance, namely, for every w it holds that $\mathsf{C}(x, w) = 0$. Fix the first message $\tilde{w}$ sent by the (cheating) prover. In particular, $\mathsf{C}(x, \tilde{w}) = 0$. Therefore, using the fact that $|\bar{x}| = \bar{n} \geq m = |w|$ (by Fact 6.7) and that E has minimal relative distance $\delta > 0$, we have that $(\bar{x}, w)$ is at least $\delta/2$ far from $(\mathsf{C}')^{-1}(1)$. Thus, by the soundness condition of Lemma 6.4, the IOPP verifier rejects with constant probability.

This concludes the analysis of our $\mathcal{C}$-encoded IOP. The full-fledged IOP of Lemma 6.6 and Theorem 3.1 now follows by applying Proposition 4.7. When applying the transformation of Proposition 4.7 we observe that the verifier in our IOP uses time $O(n)$ to create an encoding of the input x and otherwise runs in time $\mathrm{polylog}(S)$. Thus, after the application of Proposition 4.7, the verifier runs in time $O(n) + \mathrm{polylog}(S)$.

7 From fast IOPs to fast arguments

In this section we show how to compile the efficient IOP that was constructed in Section 6 into an efficient argument-system. The compiler is based on Kilian's [Kil92] PCP based construction, extended to IOPs (as proposed by [BCS16]) and using linear-time computable hash functions (see [BCG20]).

The compiler will use the following notion of a succinct commitment with local opening.

Definition 7.1. *A* succinct commitment with local opening *consists of a probabilistic algorithm called* setup *and three determinstic algorithms* commit, reveal *and* verify. *The setup algorithm, on input $n, \lambda \in \mathbb{N}$ outputs a reference string* crs. *The commit algorithm, on input* crs *and a message $m \in \{0,1\}^n$ outputs a commitment c. The algorithm* reveal, *given as input* crs, *m and an index $i \in [n]$ outputs the value $m_i \in \{0,1\}$ as well as a proof $\pi \in \{0,1\}^{\mathrm{poly}(\log n, \lambda)}$. The algorithm* verify *gets as input* crs, *c, i, b and π and outputs either "accept" or "reject".*

We require:

- *(Correctness:) For every n, λ and $m \in \{0,1\}^n$ and $i \in [n]$, it holds that* $\mathsf{verify}(\mathsf{crs}, c, i, m_i, \pi) =$ *accept, where* $\mathsf{crs} \leftarrow \mathsf{setup}(n, \lambda)$, $c = \mathsf{commit}(\mathsf{crs}, m)$ *and* $\pi \leftarrow \mathsf{reveal}(\mathsf{crs}, m, i)$.
- *(Succinctness:) The lengths of c and π are* $\mathrm{poly}(\lambda, \log n)$.
- *(Binding:) For every family of polynomial-sized circuits $\mathcal{A} = (\mathcal{A})_\lambda$, the probability over the choice of* $\mathsf{crs} \leftarrow \mathsf{setup}(n, \lambda)$ *that $A_\lambda(\mathsf{crs})$ outputs (c, i, π_0, π_1) such that* $\mathsf{verify}(\mathsf{crs}, c, i, 0, \pi_0) =$ *accept and* $\mathsf{verify}(\mathsf{crs}, c, i, 1, \pi_1) =$ *accept is negligible.*

[29] Regarding the query complexity, observe that the verifier of Lemma 6.4 has $\mathrm{polylog}(S)$ *size* and therefore makes at most that many queries.

The following lemma establishes the existence of a succinct commitment with local openings, in which the commitment can be implemented by a linear-size circuit, based on linear-time computable collision resistant hash functions. Such hash functions were constructed by Applebaum *et al.* [AHI+17].

Lemma 7.2. *Assuming that there exists a collision-resistant hash function computable by linear size circuits, with shrinkage factor* $1/2$*. Then, there exists a succinct commitment with local openings such that the commit and reveal algorithms can be implemented by circuits of size* $O(n+\lambda)$ *and the setup and verify algorithms can be implemented by Boolean circuits of size* $\mathrm{poly}(\log n, \lambda)$*.*

Proof Sketch. The construction is based on the well known Merkle tree hashing, using authentication paths in order to decommit.

In more detail, using the linear-time encodable hash function it is straightforward to commit to the Merkle root of the message in linear-time. In order to decommit to location i, one first constructs the entire Merkle tree and then selects the relevant locations from the tree (namely those blocks corresponding to the path from the i leaf in the tree to the root, together with the corresponding siblings). We use here the fact that there exists a size $O(n)$ circuit that on input an array $x \in \{0,1\}^n$ and index $i \in [n]$ outputs x_i (see Appendix D). Since we need to do two such selections (of blocks) from each layer in the tree, and the size of the layers shrinks geometrically, the overall size of the circuit is $O(n)$. □

The following lemma shows how use such linear-size CRHFs to efficiently compile an IOP into an argument-system (c.f., [BCG+17b, BCG20]).

Lemma 7.3. *Suppose that* $\mathcal{L}$ *has an* ℓ*-round* IOP *with soundness error* ε *and communication complexity* c *and that in very round either the verifier reads the prover's entire message or a constant number of bits. Suppose further that there exists a* CRHF *computable by a linear-size circuit. Then,* $\mathcal{L}$ *has an* $(\ell+2)$*-round argument-system with soundness error* $\varepsilon + \mathbf{neg}(\lambda)$ *and communication complexity* $O\big((\ell + \log c)\cdot\lambda\big)$*.*

Furthermore, suppose that:

1. *The* IOP *prover can be implemented as a size* T_P *circuit, where* $\mathrm{poly}(\lambda) \leq T_P$*.*
2. *The* IOP *verifier can be implemented in size* T_V *after a private preprocessing step of size* P_V*, where* $\mathrm{poly}(\lambda) \leq P_V$*, and that the number of queries to the preprocessed data is also* q*.*

Then, the prover of the argument-system can be implemented as a size $O(T_P)$ *circuit, and the verifier can be implemented by a size* $T_V + \mathrm{poly}(\ell, \log(T_P), \lambda)$ *circuit, following an* $O(P_V)$ *preprocessing step.*

Proof Sketch. The proof follow readily from an extension of Kilian's [Kil92] protocol to IOPs (instead of PCPs), as proposed by Ben Sasson *et al.* [BCS16]. Details follow.

The protocol involves three phases:

1. **(Preprocessing:)** The verifier runs the IOP verifier to generate the preprocessed data and then commits to the data using the succinct commitment with local openings of Lemma 7.2. The verifier preserves this commitment c_{pp} for the query and decision stage.
2. **(Interaction:)** The parties emulate the interactive phase of the IOP, except that in every round, rather than sending the (long) IOP message, the prover commits to it using the succinct

commitment with local opening. More precisely, this is done only for rounds in which the verifier does not read the entire prover message (recall that in such a case the verifier only reads a constant number of bits). Since the protocol is public-coin, the parties can continue the interaction despite the verifier not having (yet) full oracle access to the prover's messages.

3. **(Query and decision:)** In the final stage of the IOP the verifier needs to query locations in the transcript as well as the preprocessed data. In order to do so, the verifier sends these locations to the prover, who in turn decommits. The verifier also needs to find the desired (constant number of) bits from the pre-processed data, to which it only has the commitment c_{pp}. This is done by having the prover also decommit to these bits.

Completeness is immediate, and soundness follows from [Kil92, BCS16]. We proceed to analyze the complexity of the protocol.

Verifier complexity. The preprocessing step simply involves the preprocessing step of the original verifier, as well as the commitment, and can therefore be implemented by a size $O(P_V)$ circuit. The online phase is dominated by the online phase of the IOP which has complexity T_V and the verification of the decommitments which has complexity $\mathrm{poly}(\ell, q, \log(T_P), \lambda)$.

Overall the verifier's online complexity is $O(T_V) + \mathrm{poly}(\ell, q, \log(T_P), \lambda)$.

Prover complexity. Denote the length of the prover message in round i by a_i, and observe that $\sum_i a_i = O(T_P)$. The complexity of the prover is upper bounded by $O(T_P) + \sum_i O(a_i + \lambda) = O(T_P + \ell \cdot \lambda) = O(T_P)$. □

Theorem 3.2 now follows immediately by plugging in the IOP of Lemma 6.6 into Lemma 7.3.

Acknowledgments

We thank Yuval Ishai for extremely helpful discussions and comments.

References

[ACY21] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. A PCP theorem for interactive proofs. *IACR Cryptol. ePrint Arch.*, page 915, 2021.

[AHI+17] Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 7:1–7:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017.

[AM17] Benny Applebaum and Yoni Moses. Locally computable UOWHF with linear shrinkage. *J. Cryptol.*, 30(3):672–698, 2017.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 701–732, 2019.

[BCG+17a] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Interactive oracle proofs with constant rate and query complexity. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 40:1–40:15, 2017.

[BCG+17b] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 336–365. Springer, 2017.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 19–46. Springer, 2020.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 585–594, 2013.

[BCL20] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge succinct arguments with a linear-time prover. *IACR Cryptol. ePrint Arch.*, 2020:1527, 2020.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, pages 103–128, 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 1988.

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

[BIO14] Joshua Baron, Yuval Ishai, and Rafail Ostrovsky. On linear-size pseudorandom generators and hardcore functions. *Theor. Comput. Sci.*, 554:50–63, 2014.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: - new simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 699–729. Springer, 2018.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 92–122. Springer, 2021.

[BS06] Eli Ben-Sasson and Madhu Sudan. Robust locally testable codes and products of codes. *Random Structures and Algorithms*, 28(4):387–402, 2006.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1082–1090. ACM, 2019.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 90–112. ACM, 2012.

[DI14] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 169–182. ACM, 2014.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography, ITC 2021, July 23-26, 2021, Virtual Conference*, volume 199 of *LIPIcs*, pages 5:1–5:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[DSW06] Irit Dinur, Madhu Sudan, and Avi Wigderson. Robust local testability of tensor products of LDPC codes. In *proceedings of the 9th International Workshop on Randomization and Computation (RANDOM)*, pages 304–315. Springer, 2006.

[FKL+21] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-overhead zero-knowledge for RAM programs. *IACR Cryptol. ePrint Arch.*, page 979, 2021. To Appear in CCS 2021.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology -*

CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum snarks for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021. https://ia.cr/2021/1043.

[GRR18] Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 27:1–27:11, 2018.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 433–442. ACM, 2008.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[Ish20] Yuval Ishai. Zero-knowledge proofs from information-theoretic proof systems, 2020. https://zkproof.org/2020/08/12/information-theoretic-proof-systems.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732, 1992.

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

[LSTW21] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time zero-knowledge snarks for R1CS. Cryptology ePrint Archive, Report 2021/030, 2021. https://eprint.iacr.org/2021/030.

[Mei13] Or Meir. IP = PSPACE using error-correcting codes. *SIAM J. Comput.*, 42(1):380–403, 2013.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with O (1) queries. *Ann. Math. Artif. Intell*, 56(3-4):313–338, 2009.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[Plo60] Morris Plotkin. Binary codes with specified minimum distance. *IRE Trans. Inf. Theory*, 6(4):445–450, 1960.

[RR20] Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 846–857. IEEE, 2020.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *SIAM Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[Sav98] John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.

[Sho88] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 283–290, 1988.

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Inf. Theory*, 42(6):1710–1722, 1996.

[Sud01] Madhu Sudan. Algorithmic introduction to coding theory (lecture notes), 2001.

[Tha21] Justin Thaler. Proofs, arguments, and zero-knowledge, 2021. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html.

[Vid15] Michael Viderman. A combination of testability and decodability by tensor products. *Random Structures and Algorithms*, 46(3):572–598, 2015.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1074–1091. IEEE, 2021.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. *IACR Cryptol. ePrint Arch.*, page 76, 2021. To Appear in CCS 2021.

[ZWZZ20] Jiaheng Zhang, Weijie Wang, Yinuo Zhang, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. *IACR Cryptol. ePrint Arch.*, 2020:1247, 2020. To Appear in CCS 2021.

A IOP composition

In this section we prove Lemma 2.7, restated below, which gives a composition method for IOPs. As mentioned in Section 2.2.1, the use of proof composition originates in [AS98], and is articulated as a composition of a robust PCP with a PCPP in [BGH+06, DR06]. The extension to IOPs is from [BCG+17a] (see also [RR20, Lemma 8.2.] and [ACY21]). As our setting is slightly different (in particular, we care about very small factors in running times), we provide below a full proof of this lemma.

Lemma 2.7. *(Composition) Suppose that the following exist:*

- **(Outer IOP:)** *An ℓ-round (α, ε)-robust* IOP $(\mathcal{P}, \mathcal{V})$ *for a promise problem* $(\mathsf{YES}, \mathsf{NO})$.
- **(Inner IOPP:)** *An ℓ'-round q'-query α-*IOPP $(\mathcal{P}', \mathcal{V}')$ *with soundness error ε' for the language*

$$\mathcal{L}_{(\mathcal{P},\mathcal{V})} := \left\{ (x'_{\text{exp}}, x'_{\text{imp}}) \mid \mathcal{V}_2(x'_{\text{exp}}, x'_{\text{imp}}) = \texttt{accept} \right\}.$$

  Here $x'_{\text{exp}}, x'_{\text{imp}}$ are viewed as $(x_{\text{exp}}, R_1, \ldots, R_{\ell-1})$ and $(x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$, respectively, where $(x_{\text{exp}}, x_{\text{imp}}, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$ is a transcript of $(\mathcal{P}, \mathcal{V})$, and I is the query set of $\mathcal{V}$ on this transcript.

Suppose furthermore that $\mathcal{V}_1$ is s-succinct, and that $\mathcal{V}$ is t-projectable.

Then, there exists an $(\ell + \ell')$-round q'-query IOP $(\mathcal{P}'', \mathcal{V}'')$ *for the promise problem* $(\mathsf{YES}, \mathsf{NO})$ *with soundness error $\varepsilon + \varepsilon'$, prover size $|\mathcal{P}''| = |\mathcal{P}| + |\mathcal{P}'| + t$, and verifier size $|\mathcal{V}''_1| \leq |\mathcal{V}'_1| \cdot s$ and $|\mathcal{V}''_2| = |\mathcal{V}'_2|$.*

Proof. The composed protocol $(\mathcal{P}'', \mathcal{V}'')$ is given in Fig. 5.

The composed protocol $(\mathcal{P}'', \mathcal{V}'')$:

1. The prover and the verifier interact for ℓ rounds according to the outer IOP $(\mathcal{P}, \mathcal{V})$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, let $(x_{\text{exp}}, x_{\text{imp}}, M_1, R_1, \ldots, R_{\ell-1}, M_\ell)$ denote the resulting transcript.
2. The prover computes the query set $I = \mathcal{V}_1(x_{\text{exp}}, R_1, \ldots, R_{\ell-1})$, and computes the projection $(x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$.
3. The prover and the verifier interact for ℓ' rounds according to the inner IOPP $(\mathcal{P}', \mathcal{V}')$ for $\mathcal{L}_{(\mathcal{P},\mathcal{V})}$, on explicit input $x'_{\text{exp}} = (x_{\text{exp}}, R_1, \ldots, R_{\ell-1})$, and implicit input $x'_{\text{imp}} = (x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$, let $(x'_{\text{exp}}, x'_{\text{imp}}, M'_1, R'_1, \ldots, R'_{\ell'-1}, M'_{\ell'})$ denote the resulting transcript.
4. The verifier sets

$$\mathcal{V}''_2\left((x_{\text{exp}}, R_1, \ldots, R_{\ell-1}, R'_1, \ldots, R'_{\ell'-1}), y\right) = \mathcal{V}'_2\left((x'_{\text{exp}}, R'_1, \ldots, R'_{\ell'-1}), y\right),$$

   and

$$\mathcal{V}''_1(x_{\text{exp}}, R_1, \ldots, R_{\ell-1}, R'_1, \ldots, R'_{\ell'-1}) = \mathcal{V}'_1(x'_{\text{exp}}, R'_1, \ldots, R'_{\ell'-1}),$$

   where each output query location i of $\mathcal{V}'_1$ to x'_{imp} is replaced with the succinct circuit $\hat{\mathcal{V}}_1(x_{\text{exp}}, R_1, \ldots, R_{\ell-1}, i)$.

Figure 5: The Composed Protocol $(\mathcal{P}'', \mathcal{V}'')$

It can be verified that the round complexity, query complexity, and prover and verifier sizes are as stated, next we show completeness and soundness.

Completeness. Suppose that $x = (x_{\text{exp}}, x_{\text{imp}}) \in \mathsf{YES}$. Then by the guarantees of the outer IOP $(\mathcal{P}, \mathcal{V})$, with probability 1, we have that

$$\mathcal{V}_2\left((x_{\text{exp}}, R_1, \ldots, R_{\ell-1}), (x_{\text{imp}}, M_1, \ldots, M_\ell)|_I\right) = \mathcal{V}_2(x'_{\text{exp}}, x'_{\text{imp}}) = \texttt{accept}.$$

This implies in turn that $(x'_{\text{exp}}, x'_{\text{imp}}) \in \mathcal{L}_{(\mathcal{P},\mathcal{V})}$, and so by the guarantees of the inner IOPP $(\mathcal{P}', \mathcal{V}')$, the verifier $\mathcal{V}'$ will accept with probability 1. We conclude that in this case the verifier in the composed protocol accepts with probability 1.

Soundness. Suppose that $x = (x_{\text{exp}}, x_{\text{imp}}) \in \mathsf{NO}$, and let $\mathcal{P}^*$ be a prover strategy. By the robustness property of the outer IOP $(\mathcal{P}, \mathcal{V})$, with probability at least $1 - \varepsilon$, $x'_{\text{imp}} = (x_{\text{imp}}, M_1, \ldots, M_\ell)|_I$ is α-far from any y so that $\mathcal{V}_2(x'_{\text{exp}}, y) = 1$. By the guarantees of the inner IOPP $(\mathcal{P}', \mathcal{V}')$, this implies in turn that the verifier $\mathcal{V}'$ will reject with probability at least $1 - \varepsilon'$. We conclude that in this case the verifier in the composed protocol rejects with probability at least $(1 - \varepsilon) \cdot (1 - \varepsilon')$. Consequently, the soundness error is at most $1 - (1 - \varepsilon) \cdot (1 - \varepsilon') \leq \varepsilon + \varepsilon'$.

□

B Eliminating the $\mathcal{C}$-encoded assumption

In this Section we prove Proposition 4.7, restated for convenience below.

Proposition 4.7. *Suppose that $\mathcal{L}$ has an ℓ-round $\mathcal{C}$-encoded IOP $(\mathcal{P}, \mathcal{V})$ with constant soundness error. Suppose furthermore that the verifier makes a constant number of queries to the input, and for each message of $\mathcal{P}$, the verifier either reads the entire message, or makes a constant number of queries to this message.*

Then $\mathcal{L}$ also has an $(\ell + 2)$-round standard IOP $(\mathcal{P}', \mathcal{V}')$ with constant soundness error. If the prover and verifier in $(\mathcal{P}, \mathcal{V})$ have sizes $|\mathcal{P}|, |\mathcal{V}|$, respectively, then the prover and verifier in $(\mathcal{P}', \mathcal{V}')$ have sizes $O(|\mathcal{P}|)$ and $\text{poly}(|\mathcal{V}|) + \ell \cdot \text{polylog}(|\mathcal{P}|)$, respectively. Moreover, the verifier makes a constant number of queries to the input, and for each message of $\mathcal{P}'$, the verifier $\mathcal{V}'$ either reads the entire message, or makes a constant number of queries to this message.

For the proof of Proposition 4.7, we rely on the local testing and relaxed local correction properties of tensor codes. Specifically, whenever the verifier expects a codeword of $\mathcal{C}$, it first applies a local testing protocol on the prover's message to reject messages that are far from the code, and then applies a relaxed local correction protocol to decode the message to the nearest codeword.

The local testing protocol is given in the following lemma.

Lemma B.1 (Local testing protocol for tensor codes)**.** *Let $\mathbb{F}$ be a constructible finite field of characteristic 2, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 3$ and $\alpha > 0$, there exists a 2-round $\text{poly}(\delta^{-t}/\alpha)$-query IOP $(\mathcal{P}, \mathcal{V})$ with soundness error $1 - \delta^{O(t)} \cdot \alpha$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = C^{\otimes t},$$
$$\mathsf{NO} = \left\{ w \in \mathbb{F}^{n^t} \ : \ \text{dist}_{\mathbb{F}}(w, C^{\otimes t}) \geq \alpha \right\}.$$

Assuming that C can be encoded in time T, the prover has size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|)) + \text{poly}(T, t)$, and the verifier has size $\text{poly}(\log(T), \delta^{-t}/\alpha)$

We note that the above lemma could alternatively be phrased as an IOPP for membership in the code $C^{\otimes t}$ but we prefer the phrasing as an IOP to highlight that the distance is over the field $\mathbb{F}$. Lemma B.1 follows by composing the local tester for tensor codes of [Vid15] with an inner PCPP to reduce the verifier's running time and query complexity. For completeness, we provide a full proof in Appendix B.1.

The relaxed local correction protocol is given in the following lemma.

Lemma B.2 (Relaxed local correction protocol for tensor codes)**.** *Let $\mathbb{F}$ be a constructible finite field of characteristic 2, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 1$, there exists a 2-round $\mathrm{poly}(t/\delta)$-query IOP $(\mathcal{P}, \mathcal{V})$ with soundness error $1 - \frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^t$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \left\{ w \in \mathbb{F}^{n^t}, \bar{i} \in [n]^t \ : \ w \in C^{\otimes t} \right\},$$

$$\mathsf{NO} = \left\{ w \in \mathbb{F}^{n^t}, \bar{i} \in [n]^t \ : \ \mathrm{dist}_{\mathbb{F}}(w, c) < \left(\frac{\delta}{4}\right)^t \text{ for } c \in C^{\otimes t} \text{ with } w(\bar{i}) \neq c(\bar{i}) \right\}.$$

Assuming that C can be encoded in time T, the prover has size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|)) + \mathrm{poly}(T, t)$, and the verifier has size $\mathrm{poly}(\log(T), t/\delta)$.

Lemma B.2 follows by composing the relaxed local corrector for tensor codes of [GRR18, RR20] with an inner PCPP to reduce the verifier's running time and query complexity. For completeness, we provide a full proof in Appendix B.2.

Next we prove Proposition 4.7, based on the above Lemmas B.1 and B.2.

Proof of Proposition 4.7. Recall that each code C in $\mathcal{C}$ is defined as $C = \left(\widetilde{C}\right)^{\otimes t}$, where by Proposition 4.5, $\widetilde{C}$ is a systematic linear code over a finite field $\mathbb{F}$ of relative distance at least δ_0 for an absolute constant $\delta_0 > 0$. Moreover, by Proposition 4.5 and Proposition 4.6, C and $\widetilde{C}$ can be encoded in time $T := \mathrm{poly}(m)$ and $\widetilde{T} := \mathrm{poly}(m^{1/t}) = O(m)$, respectively, where m denotes the codeword length of C, and the last equality holds for a sufficiently large constant t.

The prover and the verifier $\mathcal{P}', \mathcal{V}'$ simulate the prover and the verifier $\mathcal{P}, \mathcal{V}$, with the following modifications. Suppose that $\mathcal{P}$ sends a message w that is allegedly a codeword of some code $C = (\widetilde{C})^{\otimes t} \in \mathcal{C}$. If $\mathcal{V}$ queries all of w, then $\mathcal{V}'$ does the same, and checks that w is a codeword of C. If not, then $\mathcal{V}'$ rejects and aborts. If on the other hand, $\mathcal{V}$ makes a constant number of queries to w, then $\mathcal{P}'$ and $\mathcal{V}'$ execute the protocol Π_{ltc} given in Lemma B.1 with respect to proximity parameter $\alpha = \left(\frac{\delta_0}{4}\right)^t$ on input w. If the verifier of Π_{ltc} rejects, then $\mathcal{V}'$ rejects and aborts. Then, for each query of $\mathcal{V}$ to the i-th entry of w, $\mathcal{P}'$ and $\mathcal{V}'$ execute the protocol Π_{rlcc} given in Lemma B.2 on input w and input coordinate i. If the verifier of Π_{rlcc} rejects, then $\mathcal{V}'$ rejects and aborts. In the case of a non-abort, $\mathcal{P}'$ and $\mathcal{V}'$ continue with the simulation of $(\mathcal{P}, \mathcal{V})$.

To see the correctness of $(\mathcal{P}', \mathcal{V}')$, note first that if $\mathcal{V}$ queries all of w, then we may assume that w is a codeword of C, otherwise $\mathcal{V}'$ will readily reject. If, on the other hand, $\mathcal{V}$ makes a constant number of queries to w, then we may assume that $\mathrm{dist}_{\mathbb{F}}(w, c) < \left(\frac{\delta_0}{4}\right)^t$ for some codeword c of C, as otherwise, by the soundness guarantees of Π_{ltc}, the verifier rejects with probability at least $1 - \delta_0^{O(t)} = \Omega(1)$ (assuming that t is constant). But if this is the case, then we may further assume that $w(i) = c(i)$, as otherwise, by the soundness guarantees of Π_{rlcc}, the verifier rejects with

probability at least $1 - \delta_0^{O(t)} = \Omega(1)$. Thus, in both cases we may assume that the prover sent a legal codeword of C, and correctness then follows by the $\mathcal{C}$-encoded property of $(\mathcal{P}, \mathcal{V})$.

Next we compute the prover and verifier sizes. To this end note that in the case that $\mathcal{V}$ queries all of w, the prover size is unchanged, while the verifier size is increased by an additive factor of $T := \text{poly}(m)$ that is required for checking that w is a codeword of C (by encoding the systematic part of w via C, and checking that the resulting codeword equals w). Next recall that Π_{ltc} and Π_{rlcc} have prover and verifier sizes $O(t^2 \cdot m \cdot \log(|\mathbb{F}|)) + \text{poly}(\widetilde{T}, t) = O(m \cdot \log(|\mathbb{F}|))$ and $\text{poly}(\log(\widetilde{T}), \delta_0^{-t}) \leq \text{polylog}(m)$, respectively, where the inequalities hold for a sufficiently large constant t. Consequently, in the case where $\mathcal{V}$ makes a constant number of queries to w, the prover and verifier sizes are increased by additive factors of $O(m \cdot \log(|\mathbb{F}|))$ and $\text{polylog}(m) \leq \text{polylog}(|\mathcal{P}|)$, respectively. Overall, we conclude that in the new protocol $(\mathcal{P}', \mathcal{V}')$ the prover and verifier sizes are at most $O(|\mathcal{P}|)$ and $\text{poly}(|\mathcal{V}|) + \ell \cdot \text{polylog}(|\mathcal{P}|)$, respectively.

Finally, note that since both Π_{ltc} and Π_{rlcc} are 2-round protocols, and can be executed in parallel for all queries of $\mathcal{V}$, this transformation increases the round complexity by an additive factor of 2. Moreover, since the query complexity in both Π_{ltc} and Π_{rlcc} is $\delta_0^{-O(t)} = O(1)$, we still have the property that for each message of $\mathcal{P}'$, the verifier $\mathcal{V}'$ either reads the entire message, or makes a constant number of queries to this message.

□

B.1 Local testing protocol

In this section we prove Lemma B.1 which gives a local testing protocol for tensor codes. The proof follows by composing the following local testing procedure for tensor codes from [Vid15] with an inner PCPP to reduce the verifier's running time and query complexity.

Theorem B.3 (Local testing of tensor codes, [Vid15], Theorem 3.1)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 3$, there exists a randomized oracle algorithm $\mathcal{A}$ satisfying the following properties.*

- **Input:** *$\mathcal{A}$ gets oracle access to a string $w \in \mathbb{F}^{n^t}$.*
- **Completeness:** *If w is a codeword of $C^{\otimes t}$, then $\mathcal{A}$ accepts with probability 1.*
- **Robustness:** *If $\text{dist}_{\mathbb{F}}(w, C^{\otimes t}) \geq \alpha$, then, in expectation, $\mathcal{A}$'s view is $\frac{\delta^{2t} \cdot \alpha}{t^{10}}$-far from an accepting view (where the distance is measured over $\mathbb{F}$).*
- **Running time:** *If C can be encoded in time T, then $\mathcal{A}$ has running time $O(n \cdot T)$.*

Moreover,

- *$\mathcal{A}$ queries n^2 symbols of w, and has randomness complexity $t \log n$.*
- *The location of each individual query can be computed in time $O(t \cdot \log n)$.*
- *There exists a Boolean circuit of size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$ such that given the query set I and w, outputs the query values $w|_I$.*

Remark B.4. *We remark on the following differences from [Vid15, Theorem 3.1]:*

1. *[Vid15, Theorem 3.1] only bounds the rejection probability of the n^2-query test that chooses a random two-dimensional axis-parallel plane (according to some specified distribution), and checks that the projection of w to the plane is a codeword of $C^{\otimes 2}$. However, the proof shows that this test is robust, in the sense that the average view of the tester is $\frac{\delta^{2t}\cdot\alpha}{t^{10}}$-far from $C^{\otimes 2}$.*

2. *The running time is not stated explicitly in [Vid15]. However, checking whether a given string $w' \in \mathbb{F}^{n^2}$ is a codeword of $C^{\otimes 2}$ can be done by first encoding the systematic part of w' via $C^{\otimes 2}$, and then checking that the resulting codeword equals w'. By Claim 2.14, $C^{\otimes 2}$ can be encoded in time $O(n \cdot T)$, which results in a total running time of $O(n \cdot T)$.*

3. *Randomness complexity is not stated explicitly in [Vid15]. However, inspection shows that the randomness complexity required for sampling the random two-dimensional plane is at most $t \log n$, and the location of each individual point in this subspaces can be computed in time $O(t \log n)$.*

4. *We construct a circuit for generating the projection of w to the verifier's query set as follows. First, given w, the verifier generates the restriction of w to all two-dimensional axis-parallel subspaces. Since there are $\binom{t}{2} \cdot n^{t-2}$ such subspaces, each of size $n^2 \cdot \log(|\mathbb{F}|)$ bits, this can be done by a circuit of size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$.*

   The resulting list is viewed as a database with $\binom{t}{2} \cdot n^{t-2}$ entries, each over an alphabet of bit length $n^2 \cdot \log(|\mathbb{F}|)$. The desired entry can now be selected by the multiplexer of Proposition D.1 with a size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$ circuit.

Next we prove Lemma B.1, based on the above theorem.

Proof of Lemma B.1. We apply Corollary 2.10 on the local tester given by Theorem B.3 (which is in particular also a 1-round robust IOP).

In more detail, by Theorem B.3, Markov's inequality, and our assumption that the elements of $\mathbb{F}$ are represented as codewords of a binary code of constant relative distance (cf., Section 2.3.2), there exists a 1-round $(\Omega(\frac{\delta^{2t}\cdot\alpha}{t^{10}}), 1 - \frac{\delta^{2t}\cdot\alpha}{2t^{10}})$-robust IOP $(\mathcal{P}, \mathcal{V})$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$ with prover size $|\mathcal{P}| = 0$ and verifier size $|\mathcal{V}_1| \leq \mathrm{poly}(n, t)$ and $|\mathcal{V}_2| \leq \mathrm{poly}(T)$.

Moreover, we have that:

1. $(\mathcal{P}, \mathcal{V})$ has query complexity $n^2 \cdot \log(|\mathbb{F}|)$ and randomness complexity $t \log n$.

2. The language $\mathcal{L}_{(\mathcal{P},\mathcal{V})}$ can be decided in time $\mathrm{poly}(T)$.

3. $\mathcal{V}_1$ is $\mathrm{poly}(\log n, t)$-succinct, and $\mathcal{V}$ is $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$-projectable.

Consequently, by Corollary 2.10 there exists a 2-round $\mathrm{poly}(\delta^{-t}/\alpha)$-query IOP $(\mathcal{P}', \mathcal{V}')$ for $(\mathsf{YES}, \mathsf{NO})$ with soundness error $1 - \frac{\delta^{2t}\cdot\alpha}{4t^{10}} = 1 - \delta^{O(t)} \cdot \alpha$, prover size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|)) + \mathrm{poly}(T, t)$, and verifier size $\mathrm{poly}(\log(T), \delta^{-t}/\alpha)$, where we used that $T \geq n \cdot \log(|\mathbb{F}|)$. □

B.2 Relaxed local correction protocol

In this section we prove Lemma B.2 which gives a relaxed local correction protocol for tensor codes. The proof follows by composing the following relaxed local correction procedure for tensor codes with an inner PCPP to reduce the verifier's running time and query complexity.

Lemma B.5 (Relaxed local correction of tensor codes)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 1$, there exists a randomized oracle algorithm $\mathcal{A}$ satisfying the following properties.*

- **Input:** *$\mathcal{A}$ takes as input a coordinate $\bar{i} \in [n]^t$, and also gets oracle access to a string $w \in \mathbb{F}^{n^t}$.*
- **Completeness:** *If w is a codeword of $C^{\otimes t}$, then $\mathcal{A}$ accepts with probability 1.*
- **Robustness:** *If $\mathrm{dist}_{\mathbb{F}}(w, c) < \left(\frac{\delta}{4}\right)^t$ for some codeword $c \in C^{\otimes t}$ with $c(\bar{i}) \neq w(\bar{i})$, then with probability at least $\left(\frac{\delta}{4}\right)^t$, the view of $\mathcal{A}$ is $\frac{\delta}{8t}$-far from an accepting view (where the distance is measured over $\mathbb{F}$).*
- **Running time:** *If C can be encoded in time T, then $\mathcal{A}$ has running time $O(t \cdot T)$.*

Moreover,

- *$\mathcal{A}$ queries $2tn$ symbols of w, and has randomness complexity $t^2 \log n$.*
- *The location of each individual query can be computed in time $O(t^2 \log n)$.*
- *There exists a Boolean circuit of size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$ such that given the query set I and w, outputs the query values $w|_I$.*

Remark B.6. *Usually a relaxed local corrector is defined as a randomized algorithm that is allowed to output either a symbol in $\mathbb{F}$ or a special symbol $\perp$. The completeness requirement then is that if w is a codeword of $C^{\otimes t}$, then $\mathcal{A}$ outputs $w(\bar{i})$ with probability 1, while the soundness requirement is that if w is sufficiently close to a codeword c of $C^{\otimes t}$, then $\mathcal{A}$ outputs a symbol in $\{c(\bar{i}), \perp\}$ with sufficiently high probability. Note, however, that our relaxed local corrector can be modified to satisfy these requirements, by outputting $\perp$ if it rejects, and outputting $w(\bar{i})$ otherwise. To facilitate composition, it will be more convenient for us to work with our (stronger) requirements.*

The above lemma is given in [RR20, Lemma 7.4] (which builds in turn on [GRR18, Lemma 5.5.]). As our setting is slightly different than that of [RR20], we provide a full proof in Appendix B.2.1 below. Next we prove Lemma B.2, based on Lemma B.5.

Proof of Lemma B.2. We apply Corollary 2.10 on the relaxed local corrector given by Lemma B.5 (which is in particular also a 1-round robust IOP).

In more detail, by Lemma B.5, and our assumption that the elements of $\mathbb{F}$ are represented as codewords of a binary code of constant relative distance (cf., Section 2.3.2), there exists a 1-round $\left(\Omega\left(\frac{\delta}{t}\right), 1 - \left(\frac{\delta}{4}\right)^t\right)$-robust IOP $(\mathcal{P}, \mathcal{V})$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$ with prover size $|\mathcal{P}| = 0$ and verifier size $|\mathcal{V}_1| \leq \mathrm{poly}(n, t)$ and $|\mathcal{V}_2| \leq \mathrm{poly}(T, t)$.

Moreover, we have that:

1. $(\mathcal{P}, \mathcal{V})$ has query complexity $2tn \log(|\mathbb{F}|)$ and randomness complexity $t^2 \log n$.
2. The language $\mathcal{L}_{(\mathcal{P},\mathcal{V})}$ can be decided in time $\mathrm{poly}(T, t)$.
3. $\mathcal{V}_1$ is $\mathrm{poly}(\log n, t)$-succinct, and $\mathcal{V}$ is $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$-projectable.

Consequently, by Corollary 2.10 there exists a 2-round $\mathrm{poly}(t/\delta)$-query IOP $(\mathcal{P}', \mathcal{V}')$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$ with soundness error $1 - \frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^t$, prover size $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|)) + \mathrm{poly}(T, t)$, and verifier size $\mathrm{poly}(\log(T), t/\delta)$, where we used that $T \geq n \log(|\mathbb{F}|)$. □

B.2.1 Proof of Lemma B.5

The relaxed local corrector $\mathcal{A}$ is given in Fig. 6 below.

Relaxed Local Corrector $\mathcal{A}$

Input: Coordinate $\bar{i} = (i_1, \ldots, i_t) \in [n]^t$, oracle access to $w \in \mathbb{F}^{n^t}$

1. For $\ell = 1, ..., t$:
   (a) Pick uniform random indices $r_{\ell+1}, \ldots, r_t \in [n]$.
   (b) Consider the string $w_\ell \in \mathbb{F}^n$ defined as:
   $$w_\ell(j) = w(i_1, \ldots, i_{\ell-1}, j, r_{\ell+1}, \ldots, r_t),$$
   for every $j \in [n]$. Query all points in w_ℓ, as well as n additional copies of $w_\ell(i_\ell)$.[a]
   (c) Check that w_ℓ is a codeword of C, and that all $n+1$ copies of $w_\ell(i_\ell)$ are identical; If not, reject and abort.
2. Accept and abort.

[a] These (seemingly redundant) queries are made in order to get *robustness*.

Figure 6: Relaxed Local Corrector for Tensor Codes

We start by showing completeness and robustness.

Completeness. Suppose that w is a codeword of $C^{\otimes t}$. Then by properties of tensor codes, we have that w_ℓ is a codeword of C for any $\ell = 1, \ldots, t$. Consequently, for any $\ell = 1, \ldots, t$, $\mathcal{A}$ will not reject in Step 1c. But in this case it will accept in Step 2, as required.

Robustness. Suppose that $\mathrm{dist}_{\mathbb{F}}(w, c) < \left(\frac{\delta}{4}\right)^t$ for some codeword $c \in C^{\otimes t}$ with $c(\bar{i}) \neq w(\bar{i})$. For $\ell = 1, \ldots, t+1$, let
$$\hat{c}_\ell := c(i_1, \ldots, i_{\ell-1}, \cdot, \ldots, \cdot) \in C^{\otimes(t-\ell+1)},$$
and
$$\hat{w}_\ell := w(i_1, \ldots, i_{\ell-1}, \cdot, \ldots, \cdot) \in \mathbb{F}^{n^{t-\ell+1}}.$$
Note that $\hat{w}_1 = w$, $\hat{c}_1 = c$, $\hat{w}_{t+1} = w(\bar{i})$, and $\hat{c}_{t+1} = c(\bar{i})$. Soundness relies on the following claim.

Claim B.7. *Suppose that for some $\ell \in \{1, \ldots, t\}$ it holds that $\mathrm{dist}_{\mathbb{F}}(\hat{w}_\ell, \hat{c}_\ell) < \left(\frac{\delta}{4}\right)^{t-\ell+1}$, but $\mathrm{dist}_{\mathbb{F}}(\hat{w}_{\ell+1}, \hat{c}_{\ell+1}) \geq \left(\frac{\delta}{4}\right)^{t-\ell}$. Then in the ℓ-th iteration, with probability at least $\frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^{t-\ell}$ over the choice of $r_{\ell+1}, \ldots, r_t$ in Step 1a, any $v \in \mathbb{F}^{2n}$ that is $\frac{\delta}{8}$-close to the view of $\mathcal{A}$ (where the distance is measured over $\mathbb{F}$) in Step 1b causes $\mathcal{A}$ to reject.*

Proof. Fix $\ell \in \{1, \ldots, t\}$, and suppose that $\mathrm{dist}_{\mathbb{F}}(\hat{w}_\ell, \hat{c}_\ell) < \left(\frac{\delta}{4}\right)^{t-\ell+1}$, but $\mathrm{dist}_{\mathbb{F}}(\hat{w}_{\ell+1}, \hat{c}_{\ell+1}) \geq \left(\frac{\delta}{4}\right)^{t-\ell}$. Then by assumption that $\mathrm{dist}_{\mathbb{F}}(\hat{w}_\ell, \hat{c}_\ell) < \left(\frac{\delta}{4}\right)^{t-\ell+1}$, with probability at least $1 - \frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^{t-\ell}$ over the choice of $r_{\ell+1}, \ldots, r_t$, we have that
$$\mathrm{dist}_{\mathbb{F}}(w_\ell, c_\ell) \leq \frac{\delta}{2}, \tag{9}$$

where $c_\ell := c(i_1, \ldots, i_{\ell-1}, \cdot, r_{\ell+1}, \ldots, r_t)$. On the other hand, by the assumption that $\mathrm{dist}_{\mathbb{F}}(\hat{w}_{\ell+1}, \hat{c}_{\ell+1}) \geq \left(\frac{\delta}{4}\right)^{t-\ell}$, with probability at least $\left(\frac{\delta}{4}\right)^{t-\ell}$ over the choice of $r_{\ell+1}, \ldots, r_t$, we have that

$$w_\ell(i_\ell) = w(i_1, \ldots, i_\ell, r_{\ell+1}, \ldots, r_t) \neq c(i_1, \ldots, i_\ell, r_{\ell+1}, \ldots, r_t) = c_\ell(i_\ell). \tag{10}$$

Next assume that both events (9) and (10) above hold, which happens with probability at least $\frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^{t-\ell}$. Suppose that $v \in \mathbb{F}^{2n}$ is $\frac{\delta}{8}$-close to the view of $\mathcal{A}$ in Step 1b in the ℓ-th iteration (where the distance is measured over $\mathbb{F}$), we shall show that v causes $\mathcal{A}$ to reject. Let $v = (v^{(1)}, v^{(2)})$ where $v^{(1)}, v^{(2)} \in \mathbb{F}^n$.

First observe that by our assumption that v is $\frac{\delta}{8}$-close to $\mathcal{A}$'s view, we have that $\mathrm{dist}_{\mathbb{F}}(v^{(1)}, w_\ell) \leq \frac{\delta}{4}$. By Eq. (9) and the triangle inequality, this implies in turn that $\mathrm{dist}_{\mathbb{F}}(v^{(1)}, c_\ell) \leq \frac{3\delta}{4}$. Since $c_\ell \in C$, and C has relative distance δ, we conclude that either $v^{(1)} = c_\ell$, or $v^{(1)} \notin C$. In the latter case $\mathcal{A}$ clearly rejects, and so we may assume that $v^{(1)} = c_\ell$.

Next observe that by our assumption that v is $\frac{\delta}{8}$-close to $\mathcal{A}$'s view, we also have that at least a $(1 - \frac{\delta}{4})$-fraction of the entries in $v^{(2)}$ are equal to $w_\ell(i_\ell)$. On the other hand, by our assumption Eq. (10), we have that $c_\ell(i_\ell) \neq w_\ell(i_\ell)$. We conclude that there exists an entry in $v^{(2)}$ that is not identical to $c_\ell(i_\ell) = v^{(1)}(i_\ell)$, and consequently $\mathcal{A}$ rejects on v. □

Next observe that by our assumptions that $\mathrm{dist}_{\mathbb{F}}(w, c) < \left(\frac{\delta}{4}\right)^t$ and $c(\bar{i}) \neq w(\bar{i})$, we have that $\mathrm{dist}_{\mathbb{F}}(\hat{w}_1, \hat{c}_1) < \left(\frac{\delta}{4}\right)^t$, but $\mathrm{dist}_{\mathbb{F}}(\hat{w}_{t+1}, \hat{c}_{t+1}) = 1 = \left(\frac{\delta}{4}\right)^0$. Consequently, there exists $\ell \in \{1, \ldots, t\}$ for which $\mathrm{dist}_{\mathbb{F}}(\hat{w}_\ell, \hat{c}_\ell) < \left(\frac{\delta}{4}\right)^{t-\ell+1}$, but $\mathrm{dist}_{\mathbb{F}}(\hat{w}_{\ell+1}, \hat{c}_{\ell+1}) \geq \left(\frac{\delta}{4}\right)^{t-\ell}$. But by the above Claim B.7, this implies in turn that with probability at least $\frac{1}{2} \cdot \left(\frac{\delta}{4}\right)^{t-\ell} \geq \left(\frac{\delta}{4}\right)^t$, any view that is $\frac{\delta}{8}$-close to $\mathcal{A}$'s view in Step 1b in the ℓ-th iteration, would cause it to reject. We conclude that with probability at least $\left(\frac{\delta}{4}\right)^t$, any view that is $\frac{\delta}{8t}$-close to $\mathcal{A}$'s view will cause $\mathcal{A}$ to reject.

Running time. $\mathcal{A}$ needs to check that w_ℓ is a codeword of C in Step 1c, which can be done in time T by encoding the systematic part of w_ℓ via the code C, and checking that the resulting codeword equals w_ℓ. This leads to a total running time of $O(t \cdot T)$.

Finally, we note that

- The query complexity is $2tn$ since $\mathcal{A}$ needs to query in Step 1b on the ℓ-th iteration the string $w_\ell \in \mathbb{F}^n$, as well as n additional copies of $w_\ell(i_\ell)$.
- In each iteration $\ell \in \{1, \ldots, t\}$, $\mathcal{A}$ generates $t - \ell$ random strings of length $\log n$ in Step 1a, and so the total randomness complexity is at most $t^2 \log n$. Moreover, the location of each individual query can be computed in time $O(t^2 \log n)$.
- We construct a circuit for generating the projection of w to the verifier's query set as follows. First, given w, the verifier generates the restriction of w to all axis-parallel lines. Since there are $t \cdot n^{t-1}$ such lines, each of size $n \cdot \log(|\mathbb{F}|)$, this can be done by a circuit of size $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$.

  The resulting list is viewed as a database with $t \cdot n^{t-1}$ entries, each over an alphabet of bit length $n \cdot \log(|\mathbb{F}|)$. Each desired entry (i.e., line) can now be selected by the multiplexer of Proposition D.1 with a size $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$ circuit. Since there are t such lines that are selected, the overall size is $O(t^2 \cdot n^t \cdot \log(|\mathbb{F}|))$.

C Sumcheck for rank 1 tensor coefficients

In this section we prove Lemma 5.2, restated below, which gives a sumcheck protocol for rank 1 tensor coefficients. That is, a protocol for checking that the input satisfies a linear equation, where the coefficients in the linear equation have the structure of a rank 1 tensor.

Lemma 5.2. *(Sumcheck for rank 1 tensor coefficients) Let $\mathbb{F}$ be a constructible finite field of characteristic 2, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 1$, there exists a $(t+1)$-round $\mathrm{poly}(t/\delta)$-query IOP $(\mathcal{P}, \mathcal{V})$ with soundness error $1 - \frac{1}{2} \cdot \delta^t$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle = b \right\},$$
$$\mathsf{NO} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle \neq b \right\},$$

where $b \in \mathbb{F}$, for any $i \in [t]$, Λ_i is a description of a uniform circuit that on input $w \in \mathbb{F}^k$ outputs $\langle \lambda_i, w \rangle$, and we view $(\Lambda_1, \ldots, \Lambda_t)$ as the explicit input, and $(b, C^{\otimes t}(x))$ as the implicit input.

Assuming that C can be encoded in time T, and each Λ_i is a $\mathrm{poly}(T')$-uniform circuit of size T', the prover has size $O(t \cdot n^{t-1} \cdot (T + T')) + \mathrm{poly}(T, T', t)$, and the verifier has size $\mathrm{poly}(\log(T), \log(T'), t/\delta)$.

Lemma 5.2 follows by composing the following protocol from [RR20, Lemma 7.1] with an inner PCPP to reduce the verifier running time and query complexity. As our setting is slightly different than that of [RR20], we provide below a full proof of this lemma.

Lemma C.1. *Let $\mathbb{F}$ be a constructible finite field of characteristic 2, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then for every integer $t \geq 1$, there exists a t-round $(\Omega\left(\frac{\delta}{t}\right), 1 - \delta^t)$-robust IOP $(\mathcal{P}, \mathcal{V})$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where:*

$$\mathsf{YES} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle = b \right\},$$
$$\mathsf{NO} = \left\{ (\Lambda_1, \ldots, \Lambda_t, b, C^{\otimes t}(x)) \ : \ \langle \lambda_1 \otimes \cdots \otimes \lambda_t, x \rangle \neq b \right\},$$

where $b \in \mathbb{F}$, and for any $i \in [t]$, Λ_i is a description of a uniform circuit that on input $w \in \mathbb{F}^k$ outputs $\langle \lambda_i, w \rangle$, and we view $(\Lambda_1, \ldots, \Lambda_t)$ as the explicit input, and $(b, C^{\otimes t}(x))$ as the implicit input.

Assuming that C can be encoded in time T, and each Λ_i is a $\mathrm{poly}(T')$-uniform circuit of size T', the prover has size $O(t \cdot n^{t-1} \cdot (T + T')) + \mathrm{poly}(T', t)$ and the verifier has size $\mathrm{poly}(T, T', t)$.

Moreover,

- *The query complexity is $(t+1)n\log(|\mathbb{F}|)$, and the randomness complexity is $t \log n$.*
- *$\mathcal{V}_2$ is a $\mathrm{poly}(T, T', t)$-uniform circuit of size $\mathrm{poly}(T, T', t)$.*
- *$\mathcal{V}_1$ is $\mathrm{poly}(\log(T), t)$-succinct, and $\mathcal{V}$ is $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$-projectable.*

We first show how to use Lemma C.1 to prove Lemma 5.2. Then, in Appendix C.1 we prove Lemma C.1.

Proof of Lemma 5.2. We apply Corollary 2.10 on the robust IOP given by Lemma C.1. In more detail, by Lemma C.1 there exists a t-round $(\Omega\left(\frac{\delta}{t}\right), 1 - \delta^t)$-robust IOP $(\mathcal{P}, \mathcal{V})$ for the promise problem $(\mathsf{YES}, \mathsf{NO})$ with prover size $|\mathcal{P}| = O(t \cdot n^{t-1} \cdot (T + T')) + \mathrm{poly}(T', t)$, and verifier size $|\mathcal{V}| = \mathrm{poly}(T, T', t)$.

Moreover, we have that:

1. $(\mathcal{P}, \mathcal{V})$ has query complexity $(t+1)n\log(|\mathbb{F}|)$, and randomness complexity $t \log n$.
2. The language $\mathcal{L}_{(\mathcal{P},\mathcal{V})}$ can be decided in time $\text{poly}(T, T', t)$.
3. $\mathcal{V}_1$ is $\text{poly}(\log(T), t)$-succinct, and $\mathcal{V}$ is $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$-projectable.

Consequently, by Corollary 2.10 there exists a $(t+1)$-round $\text{poly}(t/\delta)$-query IOP $(\mathcal{P}', \mathcal{V}')$ for $(\mathsf{YES}, \mathsf{NO})$ with soundness error $1 - \frac{1}{2} \cdot \delta^t$, prover size $O(t \cdot n^{t-1} \cdot (T + T')) + \text{poly}(T, T', t)$, and verifier size $\text{poly}(\log(T), \log(T'), t/\delta)$, where we used that $T \geq n \log(|\mathbb{F}|)$. □

C.1 Proof of Lemma C.1

The protocol is given in Fig. 7 below. It is similar to the standard sumcheck protocol [LFKN92, Mei13], except that the sums are weighted by the tensor coefficients.

Sumcheck

- ***Prover $\mathcal{P}$'s Input:*** For any $i \in [t]$ a uniform circuit Λ_i computing $\Lambda_i(w) = \langle \lambda_i, w \rangle$, $b \in \mathbb{F}$, and a codeword $c := C^{\otimes t}(x)$.
- ***Verifier $\mathcal{V}$'s Input:*** The same $\Lambda_1, \ldots, \Lambda_t$, and oracle access to b and c.

1. Set $b_1 \leftarrow b$, $c_1 \leftarrow c$.
2. The verifier queries n copies of b and checks that they are all identical.[a] If not, then it rejects and aborts.
3. For $\ell = 1, ..., t-1$:
   (a) Consider the string $w_\ell \in \mathbb{F}^n$ defined as:
   $$w_\ell(j) = \sum_{i_{\ell+1}, \ldots, i_t \in [k]} \lambda_{\ell+1}(i_{\ell+1}) \cdots \lambda_t(i_t) \cdot c_\ell(j, i_{\ell+1}, \ldots, i_t), \tag{11}$$
   for every $j \in [n]$. The prover computes w_ℓ and sends it to the verifier.
   (b) The verifier receives the string $\tilde{w}_\ell$, which is allegedly equal to w_ℓ. It checks that $\tilde{w}_\ell$ is a codeword of C and that $\sum_{j \in [k]} \lambda_\ell(j) \cdot \tilde{w}_\ell(j) = b_\ell$. If not, then it rejects and aborts.
   (c) The verifier sends a uniform random $r_\ell \in [n]$, and both parties set $b_{\ell+1} \leftarrow \tilde{w}_\ell(r_\ell)$ and $c_{\ell+1} \leftarrow c_\ell(r_\ell, \cdot, \ldots, \cdot) \in C^{\otimes(t-\ell)}$.
4. For $\ell = t$, the verifier explicitly checks that c_ℓ is a codeword of C, and that $\sum_{j \in [k]} \lambda_\ell(j) \cdot c_\ell(j) = b_\ell$. If not, it rejects; Otherwise, it accepts.

[a] These (seemingly redundant) queries are made in order to get *robustness* (recall that we regard b as part of the implicit input).

Figure 7: Sumcheck protocol for rank 1 tensor coefficients

We start by showing completeness and robustness. We first note the following.

Claim C.2. *For any $\ell = 1, \ldots, t-1$, we have that*

$$w_\ell(r_\ell) = \sum_{j \in [k]} \lambda_{\ell+1}(j) \cdot w_{\ell+1}(j),$$

where $w_t = c_t \in C$.

Proof. By Eq. (11) we have that,

$$\begin{aligned}\sum_{j\in[k]} \lambda_{\ell+1}(j)\cdot w_{\ell+1}(j) &= \sum_{j\in[k]} \lambda_{\ell+1}(j) \sum_{i_{\ell+2},\ldots,i_t\in[k]} \lambda_{\ell+2}(i_{\ell+2})\cdots\lambda_t(i_t)\cdot c_{\ell+1}(j,i_{\ell+2},\ldots,i_t)\\ &= \sum_{i_{\ell+1},\ldots,i_t\in[k]} \lambda_{\ell+1}(i_{\ell+1})\cdots\lambda_t(i_t)\cdot c_\ell(r_\ell,i_{\ell+1},\ldots,i_t)\\ &= w_\ell(r_\ell).\end{aligned}$$

□

Completeness. Completeness relies on the following claim.

Claim C.3. *Suppose that* $\langle \lambda_1\otimes\cdots\otimes\lambda_t, x\rangle = b$. *Then when* $\mathcal{V}$ *interacts with* $\mathcal{P}$, *for any* $\ell = 1,\ldots,t$ *it holds that*

$$\sum_{j\in[k]} \lambda_\ell(j)\cdot w_\ell(j) = b_\ell.$$

Proof. For $\ell = 1$, we have that

$$\begin{aligned}\sum_{j\in[k]} \lambda_1(j)\cdot w_1(j) &= \sum_{j\in[k]} \lambda_1(j) \sum_{i_2,\ldots,i_t\in[k]} \lambda_2(i_2)\cdots\lambda_t(i_t)\cdot c_1(j,i_2,\ldots,i_t)\\ &= \sum_{i_1,\ldots,i_t\in[k]} \lambda_1(i_1)\cdots\lambda_t(i_t)\cdot c_1(i_1,\ldots,i_t)\\ &= b_1,\end{aligned}$$

where the last equality follows by the assumption that $\langle \lambda_1\otimes\cdots\otimes\lambda_t, x\rangle = b$.

For $\ell = 2,\ldots,t$, by Claim C.2, we have that

$$\sum_{j\in[k]} \lambda_\ell(j)\cdot w_\ell(j) = w_{\ell-1}(r_{\ell-1}) = b_\ell,$$

where the last equality follows by the definition of b_ℓ. □

Next assume that $\langle \lambda_1\otimes\cdots\otimes\lambda_t, x\rangle = b$. We first claim that for any $\ell = 1,\ldots,t-1$, w_ℓ is a codeword of C. To see this, note that by the properties of tensor codes, for any fixed $i_{\ell+1},\ldots,i_t\in[k]$ we have that $c_\ell(\cdot, i_{\ell+1},\ldots i_t) = c(r_1,\ldots,r_{\ell-1},\cdot,,i_{\ell+1},\ldots i_t)$ is a codeword of C. Consequently,

$$w_\ell = \sum_{i_{\ell+1},\ldots,i_t\in[k]} \lambda_{\ell+1}(i_{\ell+1})\cdots\lambda_t(i_t)\cdot c_\ell(\cdot,i_{\ell+1},\ldots i_t)$$

is a linear combination of codewords of C, and by linearity is a codeword of C. Moreover, by Claim C.3, we have that $\sum_{j\in[k]}\lambda_\ell(j)\cdot w_\ell(j) = b_\ell$ for any $\ell = 1,\ldots,t-1$. We conclude that both verifier's checks on Step 3b will pass with probability 1.

Finally, for $\ell = t$, we certainly have that c_ℓ is a codeword of C, and by Claim C.3 we have that

$$\sum_{j\in[k]} \lambda_\ell(j)\cdot c_\ell(j) = \sum_{j\in[k]} \lambda_\ell(j)\cdot w_\ell(j) = b_\ell.$$

Consequently, the verifier's checks on Step 4 will pass with probability 1 as well.

We conclude that all tests will pass with probability 1, and so the verifier accepts with probability 1.

Robustness. Fix a deterministic prover strategy $\mathcal{P}^*$. Robustness relies on the following claims.

Claim C.4. *Suppose that in some round $\ell \in \{1, \ldots, t-1\}$, $\sum_{j\in[k]} \lambda_\ell(j) \cdot w_\ell(j) \neq b_\ell$, and both verifier's checks on Step 3b pass. Then with probability at least δ over the choice of r_ℓ it holds that $\sum_{j\in[k]} \lambda_{\ell+1}(j) \cdot w_{\ell+1}(j) \neq b_{\ell+1}$.*

Proof. By our assumption that the verifier's checks in Step 3b pass, we have that $\sum_{j\in[k]} \lambda_\ell(j) \cdot \tilde{w}_\ell(j) = b_\ell$. By our assumption that $\sum_{j\in[k]} \lambda_\ell(j) \cdot w_\ell(j) \neq b_\ell$, this implies in turn that $w_\ell \neq \tilde{w}_\ell$. Moreover, we have that both w_ℓ and $\tilde{w}_\ell$ are codewords of C, and since C has relative distance δ, these two codewords differ on at least a δ-fraction of the coordinates. But this means that with probability at least δ over the choice of r_ℓ it holds that $w_\ell(r_\ell) \neq \tilde{w}_\ell(r_\ell)$. Recalling that on the one hand $w_\ell(r_\ell) = \sum_{j\in[k]} \lambda_{\ell+1}(j) \cdot w_{\ell+1}(j)$ by Claim C.2, and on the other hand $\tilde{w}_\ell(r_\ell) = b_{\ell+1}$ by Step 3c, we conclude that in this case $\sum_{j\in[k]} \lambda_{\ell+1}(j) \cdot w_{\ell+1}(j) \neq b_{\ell+1}$. So it holds that $\sum_{j\in[k]} \lambda_{\ell+1}(j) \cdot w_{\ell+1}(j) \neq b_{\ell+1}$ with probability at least δ over the choice of r_ℓ, as stated. □

The following claim readily implies that the protocol is $(\Omega\left(\frac{\delta}{t}\right), 1-\delta^t)$-robust, recalling our assumption that the elements of $\mathbb{F}$ are represented as codewords of a binary code of constant relative distance (cf., Section 2.3.2).

Claim C.5. *Suppose that $\langle \lambda_1 \otimes \cdots \otimes \lambda_t, x\rangle \neq b$. Then with probability at least δ^{t-1}, the verifier's view is $\frac{\delta}{2(t+1)}$-far from any accepting view, where the distance is measured over $\mathbb{F}$.*

Proof. Let $v := (b^{(n)}, \tilde{w}_1, \ldots, \tilde{w}_{t-1}, c_t) \in \mathbb{F}^{(t+1)n}$ be the verifier's view, where $b^{(n)}$ denotes n concatenated copies of b. We need to show that with probability at least δ^{t-1}, any view v' so that $\mathrm{dist}_{\mathbb{F}}(v, v') < \frac{\delta}{2(t+1)}$ is rejecting. First note that if v' is an accepting view, then it must include n repeated field elements as its a prefix, followed by t codewords of C. By the triangle inequality, this implies in turn that there is at most one accepting view v' so that $\mathrm{dist}_{\mathbb{F}}(v, v') < \frac{\delta}{2(t+1)}$, and moreover, it is of the form $v' := (b^{(n)}, w'_1, \ldots, w'_{t-1}, c_t)$, where $w'_1, \ldots, w'_{t-1} \in C$. Thus, we only need to show that this view v' is rejecting with probability at least δ^{t-1}.

Next assume that v' satisfies all verifier's checks on Step 3b, we shall show that the verifier will reject on Step 4 with probability at least δ^{t-1}. To see this, note that by our assumption that $\langle \lambda_1 \otimes \cdots \otimes \lambda_t, x\rangle \neq b$, we have that

$$\begin{aligned}
\sum_{j\in[k]} \lambda_1(j) \cdot w_1(j) &= \sum_{j\in[k]} \lambda_1(j) \sum_{i_2,\ldots,i_t\in[k]} \lambda_2(i_2) \cdots \lambda_t(i_t) \cdot c_1(j, i_2, \ldots, i_t) \\
&= \sum_{i_1,\ldots,i_t\in[k]} \lambda_1(i_1) \cdots \lambda_t(i_t) \cdot c_1(i_1, \ldots, i_t) \\
&\neq b_1.
\end{aligned}$$

Consequently, by Claim C.4, assuming that v' satisfies all verifier's checks on Step 3b, with probability at least δ over the choice of r_1 we have that $\sum_{j\in[k]} \lambda_2(j) \cdot w_2(j) \neq b_2$. If this is the case, then with probability at least δ over the choice of r_2, we have that $\sum_{j\in[k]} \lambda_3(j) \cdot w_3(j) \neq b_3$. Continuing this way, we have that $\sum_{j\in[k]} \lambda_t(j) \cdot w_t(j) \neq b_t$ with probability at least δ^{t-1}. But recalling that $w_t = c_t$, this implies in turn that $\sum_{j\in[k]} \lambda_t(j) \cdot c_t(j) \neq b_t$. We conclude that the verifier will reject in Step 4 with probability at least δ^{t-1}. □

Verifier size. We start by computing the size of $\mathcal{V}_1$. For this, first note that the verifier first queries n copies of $b \in \mathbb{F}$, then the entire prover's messages $\tilde{w}_1, \ldots, \tilde{w}_{t-1} \in \mathbb{F}^n$, and finally the entire codeword $c_t \in \mathbb{F}^n$. Hence the total query complexity is $q := (t+1)n\log(|\mathbb{F}|)$. Moreover, as the transcript has length $n^t \cdot \log(|\mathbb{F}|) + O(tn\log(|\mathbb{F}|))$, there exists a circuit of size $\mathrm{poly}(\log(n\log(|\mathbb{F}|)), t)$ that given $r_1, \ldots, r_{t-1}$ and $i \in [q]$, outputs the location of the i-th query. We conclude that $\mathcal{V}_1$ is a $\mathrm{poly}(\log(n\log(|\mathbb{F}|)), t)$-succinct circuit of size $\mathrm{poly}(n, \log(|\mathbb{F}|), t)$. Noting that $T \geq n\log(|\mathbb{F}|)$, this implies in turn that $\mathcal{V}_1$ is a $\mathrm{poly}(\log(T), t)$-succinct circuit of size $\mathrm{poly}(T, t)$.

Next we compute the size of $\mathcal{V}_2$. In each round, the verifier needs to check that $\tilde{w}_\ell$ is a codeword of C, and that $\sum_{j\in[k]} \lambda_\ell(j) \cdot \tilde{w}_\ell(j) = b_\ell$. For the first task, the verifier can simply encode the first k entries of $\tilde{w}_\ell$ via the code C and check that the resulting codeword equals $\tilde{w}_\ell$, which can be done in time T. The second task requires applying Λ_ℓ which takes time $\mathrm{poly}(T')$. Hence $\mathcal{V}_2$ is a $\mathrm{poly}(T, T', t)$-uniform circuit of size $\mathrm{poly}(T, T', t)$.

Prover size. In each round, the prover needs to compute w_ℓ. For each $j \in [n]$, the coordinate $w_\ell(j)$ can be computed by first applying the circuit Λ_t for $k^{t-\ell-1}$ times (for each value of $i_{\ell+1}, \ldots, i_{t-1} \in [k]$), then applying the circuit Λ_{t-1} for $k^{t-\ell-2}$ times (for each value of $i_{\ell+1}, \ldots, i_{t-2} \in [k]$) and so on, till finally applying the circuit $\Lambda_{\ell+1}$ for a single time. So if each circuit Λ_i has size T', then for any $\ell \in \{1, \ldots, t\}$ and $j \in [n]$, $w_\ell(j)$ can be computed by a circuit of size at most $(1 + k + \cdots + k^{t-\ell-1}) \cdot T' = O(k^{t-\ell-1} \cdot T') \leq O(k^{t-2} \cdot T')$. We conclude that the whole vector w_ℓ can be computed using a circuit of size $O(n \cdot k^{t-2} \cdot T')$. The prover also needs to generate the circuits $\Lambda_1, \ldots, \Lambda_t$ which takes time $\mathrm{poly}(T')$. We conclude that Step 3a can be preformed with prover size $O(t \cdot n \cdot k^{t-2} \cdot T') + \mathrm{poly}(T', t)$.

Additionally, in Step 3c the prover needs to compute the restriction of c_ℓ to the hyperplane $c_\ell(r_\ell, \cdot, \ldots, \cdot)$. One can construct a circuit of size $O(n^{t-\ell+1} \cdot \log(|\mathbb{F}|))$ that outputs the restriction of $c_\ell \in C^{\otimes(t-\ell+1)}$ to the hyperplane as follows. The circuit first generates the restriction of c_ℓ to all n hyperplanes. Then, the circuit uses the linear-size multiplexer of Proposition D.1 to output the specified hyperplane. Thus, the size of the circuit is at most $O(n^{t-\ell+1} \cdot \log(|\mathbb{F}|))$. Consequently, Step 3c can be performed with prover size at most $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$

We conclude that the overall prover size is $O(t \cdot n^{t-1} \cdot (T + T')) + \mathrm{poly}(T', t)$, where we used that $n\log(|\mathbb{F}|) \leq T$ and $k \leq n$.

Finally, we note that

- The query complexity is $(t+1)n\log(|\mathbb{F}|)$ since the verifier first queries n copies of $b \in \mathbb{F}$, then the entire prover's messages $\tilde{w}_1, \ldots, \tilde{w}_{t-1} \in \mathbb{F}^n$, and finally the entire codeword $c_t \in \mathbb{F}^n$
- In each round the verifier sends a uniform random index in $[n]$, and so the total randomness complexity is $(t-1)\log n$.
- Clearly, there is a circuit of size $O(n^t \cdot \log(|\mathbb{F}|))$ that given the transcript, outputs n copies of b, as well as the entire prover's messages $\tilde{w}_1, \ldots, \tilde{w}_{t-1}$. Moreover, by the discussion above (see 'Prover size' paragraph) there is a circuit of size $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$ that given the transcript, outputs c_t. We conclude that the verifier is $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$-projectable.

D Linear-size selection

We construct a linear-size circuit, that given a string $x \in \Sigma^n$, over an alphabet Σ, and an index $i \in [n]$, outputs the symbol x_i (in other words, a multiplexer).

Proposition D.1. *There exists a size $O(n \cdot \log(|\Sigma|))$ circuit that given as input $x \in \Sigma^n$ and $i \in [n]$ outputs x_i.*

A proof of Proposition D.1 appears in [Sav98, Lemma 2.5.5]. We give here a sketch for a simple alternate construction.

Proof Sketch. The circuit follows a divide and conquer strategy. Assume for simplicity that n is a power of 2. Let $x \in \Sigma^n$ and $i \in [n]$ be the inputs. Denote by σ the MSB of i and by i' the remaining bits (i.e., $i' \in [n/2]$). To select the i-th index of x, the circuit recursively selects the i' index from both the lower and upper halves of x. Then, based on σ, it decides which of the two bits to output. Overall the circuit size is:

$$S(n) = 2S(n/2) + O(\log(|\Sigma|)) = \cdots = O(n \cdot \log(|\Sigma|)),$$

$$S(1) = O(\log(|\Sigma|)).$$

□

E Linear-size evading set generator

In this section we prove Proposition 2.22, restated below.

Proposition 2.22. *(Linear-size evading set generator) There exists $d = O(\log n)$, and an 0.49-evading set generator $S : \{0,1\}^d \to \{0,1\}^n$ that can be evaluated by a size $O(n)$ Boolean circuit.*

We begin with a couple of basic claims. The first claim gives a way to extend the output length of the generator, while incurring some loss in ε.

Claim E.1. *If $S : \{0,1\}^d \to \{0,1\}^n$, $S' : \{0,1\}^{d'} \to \{0,1\}^{n'}$ are $\varepsilon, \varepsilon'$-evading set generators, respectively, then $S \otimes S' : \{0,1\}^{d+d'} \to \{0,1\}^{n \cdot n'}$ is an $(\varepsilon \cdot \varepsilon')$-evading set generator, where*

$$(S \otimes S')(x, x') = S(x) \otimes S'(x'),$$

for $x \in \{0,1\}^d, x' \in \{0,1\}^{d'}$.

Proof. Let y be a non-zero vector in $\{0,1\}^{n \cdot n'}$. We use $Y \in \{0,1\}^{n \times n'}$ to denote the matrix obtained by viewing y as an $n \times n'$ dimensional matrix in the natural way. Then,

$$\left\langle (S \otimes S')(x, x'), y \right\rangle \left\langle S(x) \otimes S'(x'), y \right\rangle = \left\langle S(x), Y \cdot S'(x') \right\rangle.$$

Since Y is non-zero, it has some non-zero row Y_i. Since S' is ε'-evading, with probability at least ε' it holds that $\langle Y_i, S'(x') \rangle \neq 0$. Assuming that this event holds, $Y \cdot S'(x')$ is a non-zero vector and so since S is ε-evading, $\langle S(x), Y \cdot S'(x') \rangle \neq 0$ with probability at least ε. Overall we get that $\langle (S \otimes S')(x, x'), y \rangle$ is non-zero with probability at least $\varepsilon \cdot \varepsilon'$. □

In what follows, for an evading set generator S we let $S^{\otimes 1} := S$, and $S^{\otimes t} := S^{\otimes t-1} \otimes S$ for any $t \geq 2$. The next claim gives a way to amplify the value ε of an evading set generator.

Claim E.2. *If $S : \{0,1\}^d \to \{0,1\}^n$, $S' : \{0,1\}^s \to \{0,1\}^t$, are $\varepsilon, \varepsilon'$-evading set generators, respectively, then $S \circ S' : \{0,1\}^{t \cdot d + s} \to \{0,1\}^n$ is an $\varepsilon' \cdot (1 - (1-\varepsilon)^t)$-evading set generator, where*

$$(S \circ S')(x_1, \ldots, x_t, z) = (S'(z))_1 \cdot S(x_1) + (S'(z))_2 \cdot S(x_2) + \cdots + (S'(z))_t \cdot S(x_t),$$

for $x_1, \ldots, x_t \in \{0,1\}^d$, $z \in \{0,1\}^s$, and where addition is performed over $\mathbb{GF}(2)$.

Proof. Let y be a non-zero vector in $\{0,1\}^n$. Then,

$$\begin{aligned} \left\langle (S \circ S')(x_1, \ldots, x_t, z), y \right\rangle &= \left\langle (S'(z))_1 \cdot S(x_1) + \cdots + (S'(z))_t \cdot S(x_t), y \right\rangle \\ &= \left\langle S'(z), \left(\langle S(x_1), y \rangle, \ldots, \langle S(x_t), y \rangle \right) \right\rangle. \end{aligned}$$

Since S is ε-evading, with probability at least $1 - (1-\varepsilon)^t$ it holds that $\langle S(x_i), y \rangle \neq 0$ for some $i \in [t]$. If this event holds, then $(\langle S(x_1), y \rangle, \ldots, \langle S(x_t), y \rangle)$ is a non-zero vector, and so since S' is ε' evading, we have that $\langle S'(z), (\langle S(x_1), y \rangle, \ldots, \langle S(x_t), y \rangle) \rangle$ is non-zero with probability at least ε'. Overall we get that $\langle (S \circ S')(x_1, \ldots, x_t, z), y \rangle$ is non-zero with probability at least $\varepsilon' \cdot (1 - (1-\varepsilon)^t)$. □

Finally, we shall make use of the following simple code-based construction of evading sets.

Claim E.3 ([NN93])**.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code of relative distance δ, and let $G \in \{0,1\}^{n \times k}$ be the generating matrix of C. Then $S : [n] \to \{0,1\}^k$ is a δ-evading set generator, where $S(i)$ is the i-th row of G.*

Proof. Let y be a non-zero vector in $\{0,1\}^k$. Then, $\langle S(i), y \rangle = (G \cdot y)_i = (C(y))_i$. Since C has relative distance δ, with probability at least δ over the choice of $i \in [n]$ we have that $(C(y))_i$ is non-zero, and so $\langle S(i), y \rangle$ is non-zero with probability at least δ. □

We now proceed to the proof of Proposition 2.22.

Proof of Proposition 2.22. We first note that it suffices to construct a linear-size ε-evading set generator $S : \{0,1\}^d \to \{0,1\}^n$ with $d = O(\log n)$ for *some* constant $\varepsilon > 0$. Indeed, given such an evading set generator we can increase ε to 0.49 by applying Claim E.2 with $t = O(1/\varepsilon)$, and $S' : \{0,1\}^{O(\log t)} \to \{0,1\}^t$ being the 0.499-evading set generator given by Claim E.3 (for any binary linear code of polynomial length and relative distance 0.499).

Let $\{C_k\}_{k \in \mathbb{N}}$ be an explicit family of binary linear codes of polynomial length and constant relative distance $\delta > 0$, where C_k has message length k. Assume furthermore that the generating matrix G_k of C_k can be computed by an $O(k^{\bar{c}})$-size circuit for an absolute constant $\bar{c}$. Let $\tilde{C} := C_{n^{1/\bar{c}}}$ (i.e., the code corresponding to message length $n^{1/\bar{c}}$), and note that the generating matrix $\tilde{G}$ for $\tilde{C}$ can be computed by an $O(n)$-size circuit. Let $\tilde{S} : \{0,1\}^{O(\log n)} \to \{0,1\}^{n^{1/\bar{c}}}$ be the δ-evading set generator guaranteed by Claim E.3 for the code $\tilde{C}$, and let $S = (\tilde{S})^{\otimes \bar{c}}$.

By Claim E.1, we have that $S : \{0,1\}^d \to \{0,1\}^n$ is an $\varepsilon := \delta^{\bar{c}} > 0$-evading set generator for $d = O(\log n)$. Next we show that S can be evaluated by an $O(n)$-size circuit. To this end, observe that by definition, S receives as input $\bar{c}$ random row indices in the generating matrix $\tilde{G}$ of $\tilde{C}$, and outputs the tensor product of the corresponding rows. The circuit computing S first computes $\tilde{G}$ which, as noted above, can be done with a size $O(n)$. Given $\tilde{G}$, each of the $\bar{c}$ rows can be selected by the multiplexer of Proposition D.1 with $O(n)$ size. Finally, since each of the $\bar{c}$ rows has size $n^{1/\bar{c}}$, their tensor product can clearly be computed with $O(n)$ size. Overall S can be evaluated by an $O(n)$-size circuit. □