Faster Sounder Succinct Arguments and IOPs

Justin Holmgren*   Ron D. Rothblum†

June 23, 2022

Abstract

Succinct arguments allow a prover to convince a verifier that a given statement is true, using an extremely short proof. A major bottleneck that has been the focus of a large body of work is in reducing the overhead incurred by the prover in order to prove correctness of the computation. By overhead we refer to the cost of proving correctness, divided by the cost of the original computation.

In this work, for a large class of Boolean circuits $C = C(x, w)$, we construct succinct arguments for the language $\{x : \exists w\; C(x, w) = 1\}$, with $2^{-\lambda}$ soundness error, and with prover overhead $\text{polylog}(\lambda)$. This result relies on the existence of (sub-exponentially secure) linear-size computable collision-resistant hash functions. The class of Boolean circuits that we can handle includes circuits with a repeated substructure, which arise in natural applications such as batch computation/verification, hashing and related block chain applications.

The succinct argument is obtained by constructing *interactive oracle proofs* for the same class of languages, with $\text{polylog}(\lambda)$ prover overhead, and soundness error $2^{-\lambda}$. Prior to our work, the best IOPs for Boolean circuits either had prover overhead of $\text{polylog}(|C|)$ based on efficient PCPs due to Ben Sasson *et al.* (STOC, 2013) or $\text{poly}(\lambda)$ due to Rothblum and Ron-Zewi (STOC, 2022).

---

*NTT Research, USA. Email: justin.holmgren@ntt-research.com.

†Technion, Israel. Email: rothblum@cs.technion.ac.il.

Contents

1 Introduction

Succinct arguments are interactive proof systems that allow a prover to convince a verifier that a computational statement is true, using an extremely short proof. Soundness is computational — no polynomial-time cheating prover can convince the verifier to accept a false statement except with negligible probability.

Succinct arguments, especially those that are *zero-knowledge* [GMR89], originate in the pioneering theoretical works of Kilian [Kil92] and Micali [Mic00]. In recent years however they have been drawing immense interest also in practice and several different systems are being developed and deployed.[1] One of the major bottlenecks to more widespread deployment is the overhead incurred by the prover — the cost of proving that a statement is true is still orders of magnitude larger than directly checking that the statement is true.

The original work of Kilian [Kil92], based on PCPs and collision resistant hash functions, has a prover that has a large polynomial overhead. Since Kilian's original work, and especially in recent years, there has been significant effort in improving the prover's runtime. In particular, a recent line of works have achieved succinct arguments with a linear-size prover [BCG+17, XZZ+19, ZWZZ20, BCG20, BCL20, LSTW21, GLS+21] for arithmetic circuits over large finite fields. Even more recently, Ron-Zewi and Rothblum [RR21] constructed succinct arguments with a *strictly linear-size prover* for general Boolean circuits. However, the soundness error in their protocol is constant, rather than negligible as we would typically desire. All of these results fall short of the holy-grail in the field, which is captured by the following question:

> *Can we construct succinct arguments and interactive oracle proofs for size-S circuits with an $O(S) + \mathrm{poly}(\lambda)$ size prover and soundness error $2^{-\lambda}$?*

We emphasize that straightforward repetition, or working over 2^λ-size finite fields, yields an $O(S \cdot \lambda)$-size prover (when implemented as a Boolean circuit). The challenge that we are faced with is therefore breaking the multiplicative dependence between the circuit size and the security parameter into an additive one.

1.1 Our Results

In this work we construct succinct arguments that come close to resolving the above question, for a large class of Boolean circuits. Our first main result is a succinct argument-system with a $|C| \cdot \mathrm{polylog}(\lambda) + \mathrm{poly}(\lambda, \log|C|)$-size prover, for the relevant class of circuits C. This result relies on the existence of linear-size computable hash functions such as those constructed by Applebaum *et al.* [AHI+17].

Theorem 1.1 (Informally Stated, see Theorem 4.6)**.** *Assume the existence of sub-exponentially secure linear-size computable hash functions.*

Then, for any Boolean circuit $C : \{0,1\}^{n+m} \to \{0,1\}$ of size S with a "nice" succinct description of size s, there exists a succinct public-coin argument for the language $\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m, \ C(x,w) = 1\}$, with $2^{-\lambda}$ soundness error and an $S \cdot \mathrm{polylog}(\lambda) + \mathrm{poly}(\lambda, \log S)$ size prover. The communication complexity is $\mathrm{poly}(\lambda, \log S)$, the number of rounds is $O(\log S)$ and the verifier runs in time $O(n + s \cdot \lambda)$.

We emphasize that the main novelty in Theorem 4.6 is that the prover has size roughly $S \cdot \mathrm{polylog}(\lambda)$, rather than $S \cdot \mathrm{poly}(\lambda)$. The "nice" class of circuits that we handle generalizes (modulo minor technicalities[2]) the notion of Succinct R1CS, introduced by Ben Sasson *et al.* [BCG+19]. Loosely speaking, this class captures computations that have some repeated sub-structure. As the precise definition is somewhat involved and quite general (see Definition 4.2) we highlight two particular examples of interest. The first is "T-iterated" circuits for $T \geq \lambda$, i.e. those which map $z = (x, w)$ to $\underbrace{(D \circ \cdots \circ D)}_{T \text{ times}}(z)$ for a small circuit D. The second is "batch" circuits, which map $(z_1, \ldots, z_T)$ to $D(z_1) \wedge \cdots \wedge D(z_T)$, again for $T \geq \lambda$. In both cases, for any $\varepsilon > 0$, we obtain a protocol where our prover has size $T \cdot |D| \cdot \mathrm{polylog}(\lambda)$ and our verifier has size $(|D| + T^\varepsilon) \cdot \mathrm{poly}(\lambda)$.

---

[1]See https://zkproof.org for additional details and resources as well as the recent surveys [Ish20, Tha21].

[2]Succinct R1CS was defined as a constraint system involving two types of constraints: time constraints and boundary constraints. We can always handle the time constraints, and handle natural boundary constraints, which were the motivation for the succinct R1CS definition.

We remark that these class of computations arise in natural scenarios involving cryptographic hashing and blockchains.

Following a body of recent works, Theorem 1.1 follows (non-trivially) from an analogous (unconditional) *interactive oracle proof* (IOP) [BCS16, RRR21]. An IOP can be thought of as an interactive version of a PCP — the prover can interact with the verifier, who in turn is allowed to read a few bits from each message sent by the prover. Our main technical result is a new efficient IOP construction for the same class of problems.

Theorem 1.2 (Informally Stated, see Theorem 4.5). *For the same family of Boolean circuits $C : \{0,1\}^{n+m} \to \{0,1\}$ of size S with a "nice" succinct description of size s, there exists an* IOP *for the language $\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m, \ C(x, w) = 1\}$, with $2^{-\lambda}$ soundness error and an $S \cdot \text{polylog}(\lambda)$ size prover. The number of rounds is $O(\log S)$, the query complexity is $s \cdot \text{poly}(\lambda)$ and the verifier runs in time $n \cdot \text{polylog}(\lambda) + s \cdot \text{poly}(\lambda)$.*

We note that there are two aspects that make the compilation of the IOP of Theorem 1.2 into the succinct argument of Theorem 1.1 non-trivial. The first is the fact that the query complexity in Theorem 1.2 has an s dependence which may be only slightly sublinear in S, whereas the communication complexity in Theorem 1.1 has a poly-logarithmic dependence on S. This improvement is actually relatively easy to achieve — we first construct an argument-system in which the communication complexity is $s \cdot \text{poly}(\lambda)$ but then compose with an off-the-shelf argument-system (e.g., the original [Kil92] argument) to reduce the communication to be poly-logarithmic. (We remark that we leave open the question of improving the verification time and query complexity to be poly-logarithmic in the IOP of Theorem 1.2.)

A more subtle, and serious, issue is that in order to implement the standard transformation of IOPs into succinct arguments ([Kil92, BCS16]) the prover needs to be able to *project* its IOP messages to the specific verifier query locations. The straightforward circuit for projecting a string of length $N = S \cdot \text{polylog}(\lambda)$ to q coordinates, has size $O(N \cdot q)$ which we cannot afford. To the best of our knowledge no circuit of size $O(N) + \text{poly}(q, \log N)$ is known for the problem, which poses a serious difficulty. In [RR21] this problem was overcome by ensuring that the verifier makes only a constant number of queries to each message (or reads it entirely). In our IOP since we are aiming for $2^{-\lambda}$ soundness error, intuitively, the verifier has to make $\Omega(\lambda)$ queries and so we cannot follow the [RR21] approach. Rather, we overcome the difficulty by ensuring a utilizing a particular query structure of our IOP verifier, see Section 2 for details.

1.2 Related Work

The general question of constructing cryptographic primitives with constant overhead was raised by Ishai *et al.* [IKOS08]. In particular, they asked whether we can construct zero-knowledge proofs (with negligible soundness error) and constant computational overhead for the prover.

As was previously mentioned, a recent exciting line of work [BCG+17, XZZ+19, ZWZZ20, BCG20, BCL20, LSTW21, GLS+21] has constructed succinct arguments for *arithmetic* circuits over large finite fields, with a linear-size prover (see also [GLS+21] for a more through discussion and comparison of some of these works). In the Boolean circuit regime, the best result is the recent work of [RR21] which achieves a linear-size prover, albeit with only a constant soundness error.

A separate line of work has focused on constructing zero-knowledge proofs with a linear-size prover, but where the *communication* may also grow linearly in the circuit size. (Here the aspect that makes the problem non-trivial is simply that the proof should be zero-knowledge.) Such a non-succinct zero-knowledge proof, with a linear-size prover, can be derived fairly directly from [IKOS09], using linear-size computable commitments, but the resulting proof-system has a constant soundness error.

Damgård *et al.* [DIK10] similarly construct non-succinct zero-knowledge proofs with comparable prover size to ours - namely, $|C| \cdot \text{polylog}(\lambda)$. We emphasize however that the [DIK10] protocol is not succinct.

Recent works by Weng *et al.* [WYKW21] and Franzese *et al.* [FKL+21] also 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.

A separate line of work focuses on the *space efficiency* of the prover. Proof-systems achieving time *and space* efficient provers are known in the designated verifier setting [BC12, HR18] as well as in the publicly verifiable setting [BHR+20, BHR+21]. The work of [EFKP20] achieves highly efficient *parallel time*.

1.3 Organization

In Section 2 we give an overview of our techniques. In Section 3 we introduce notations and definitions that will be used throughout this work. Our main results are formally stated in Section 4. In Section 5 we introduce some tools that we be used to construct our IOPs. In Section 6 we give the key IOP sub-protocols that are used in our construction. In Section 7 we combine all of the above to prove our main results. Due to space restriction, the (by now standard) definition of IOPs is given in Appendix A and, in Appendix B we highlight concrete realizations of our tensor circuit framework.

2 Technical Overview

Multi-sumcheck with Small Error. The starting point of our work is the aforementioned recent work of Ron-Zewi and Rothblum [RR21]. One of the key technical ideas in that work is an efficient "multi-sumcheck" protocol. Generally speaking, a *multi-sumcheck* IOP is an IOP in which the verifier is given oracle access to a pair of codewords c, c', belonging to a code $C : \mathbb{F}^k \to \mathbb{F}^n$, and would like to compute the inner product $\sum_{i \in [k]} c(i) \cdot c'(i)$.[3] Ron-Zewi and Rothblum construct a linear-size encodable code C which has a multi-sumcheck protocol with a linear-size prover. Unfortunately, the protocol only has a constant soundness error.

We first discuss two common approaches for error reduction. The first is simply to repeat the protocol $O(\lambda)$ times. This indeed reduces the soundness error at an exponential rate, but naturally increases the prover's size by a λ multiplicative factor, which we would like to avoid. Another common approach is to try to work with codes with very large minimal distance, say $1 - 2^{-\lambda}$. Unfortunately, by the Plotkin bound, such codes require an exponentially-large alphabet which would again introduce a poly(λ) multiplicative factor in runtime.

Thus, our first key insight is a (simple) method for reducing the soundness error of the [RR21] protocol to $2^{-\lambda}$, but with only a polylog(λ) overhead in the prover's size.

Let $\mathbb{F}$ be a finite field of size $O(\lambda)$ and consider the Reed-Solomon code $RS_\lambda : \mathbb{F}^\lambda \to \mathbb{F}^{O(\lambda)}$ over $\mathbb{F}$ - namely, the code consisting of all degree $\lambda - 1$ polynomials over $|\mathbb{F}|$. We will use two key properties of the Reed-Solomon code: (1) that it is a *multiplication code* (since the point-wise (aka Hadamard) product of any two polynomials is a polynomial of degree at most 2λ and therefore belongs to a closely related Reed-Solomon code) *and* (2) that it can be encoded by a size $\lambda \cdot$ polylog(λ) circuit (using the Fast Fourier Transform). We remark that the parameters (in particular the field size and block length) are set so that RS_λ has a constant relative distance and further note that we could replace the Reed-Solomon code with any constant-distance multiplication code with quasi-linear time encoding.

In addition to the ubiquitous Reed-Solomon code, we will also use the code C from [RR21]. Indeed, we will combine these two codes to construct a new code D and show an efficient multi-sumcheck procedure for D with a small soundness error.

The code D is simply the tensor product of RS_λ with C, denoted $D = C \otimes RS_\lambda$. In this code, messages are viewed as $(k/\lambda) \times \lambda$ matrices and we encode them by encoding first the rows using RS_λ and encode the columns (both old and the new ones generated by the row encoding) using C (where we use C with respect to message size k/λ).[4] Observe that since RS_λ is encodable by a $\lambda \cdot$ polylog(λ)-size circuit, and C is encodable by a linear-size circuit, the code D is overall encodable by a circuit of size $(k/\lambda) \cdot \lambda \cdot$ polylog(λ) $+ O(\lambda) \cdot O(k/\lambda) =$

---

[3]For simplicity we focus here on the case of two codewords, but in general we would like to be able to handle any constant number of codewords.

[4]A minor technical inaccuracy is that the Reed-Solomon code's alphabet is $\mathbb{F}$ whereas the [RR21] code has a binary alphabet. This can be easily be resolved by a simple extension of the [RR21] result to the larger alphabet size, while accounting for additional polylog($|\mathbb{F}|$) factors in efficiency, which we can afford since $|\mathbb{F}| = O(\lambda)$.

$k \cdot \mathrm{polylog}(\lambda)$. Thus, the code D maps messages of length k to codewords of length $k \cdot \mathrm{polylog}(\lambda)$ and is encodable by a $k \cdot \mathrm{polylog}(\lambda)$ size circuit.

Our first main step is showing a multi-sumcheck protocol for D, with soundness error $2^{-\lambda}$. Recall that the input to this protocol is two codewords $d, d' \in D$, and we would like to check that $\sum_{i \in [k/\lambda], j \in [\lambda]} d(i, j) \cdot d'(i, j) = b$, for some scalar b. The protocol is simple - the prover generates the codeword $d_{sum} \in RS_{2\lambda}$ (we use $R_{2\lambda}$ to denote the Reed-Solomon code of the same block length as RS_λ but double the degree) defined as $d_{sum} = \sum_{i \in [k]} d_i \star d'_i$, where d_i and d'_i denote the i-th rows of d and d', respectively, and $\star$ denotes a point-wise/Hadamard product. Note that by linearity, d_{sum} is indeed a codeword of $RS_{2\lambda}$ and that $\sum_j d_{sum}(j) = \sum_{i,j} d(i, j) \cdot d'(i, j)$. The prover computes d_{sum} and sends it explicitly to the verifier. The verifier in turn, after receiving the message $\tilde{d}_{sum}$ (which may or may not be equal to the intended d_{sum}) checks that $\tilde{d}_{sum} \in RS_{2\lambda}$ and that $\sum_{j \in [\lambda]} \tilde{d}_{sum}(j) = b$.

Thus, if the original claim is false, in order not to be caught already at this stage, the prover must send a false codeword $\tilde{d}_{sum} \not\equiv d_{sum}$.

At this point we observe that $\tilde{d}_{sum}$ and d_{sum} are distinct codewords and so they must disagree on a constant fraction of their coordinates. A typical "sum-check" approach would be for the verifier to choose at random one of these coordinates and recursively check the resulting multi-sumcheck claim. This is the approach taken both in the classical sum-check as well as in [RR21].

Our approach differs here. In a nutshell, we observe that we can afford to run the [RR21] multi-sumcheck procedure for *each and every one* of the claims induced by $\tilde{d}_{sum}$, each with a constant soundness error, say $1/2$. Since *many* (i.e., $\Omega(\lambda)$) of the claims are false, the probability that the prover will successfully cheat in *all* of these invocation is $2^{-\Omega(\lambda)}$.

In more detail, denote the set of coordinates on which d_{sum} and $\tilde{d}_{sum}$ differ by $J \subseteq [O(\lambda)]$. That is,

$$J = \left\{ j : \tilde{d}_{sum}(j) \neq d_{sum}(j) = \sum_{i \in [k/\lambda]} d(i, j) \cdot d'(i, j) \right\}.$$

Recall that by the above arguments, we know that $|J| = \Omega(\lambda)$. At this point we run the [RR21] multi-sumcheck protocol, with a constant soundness error, on each and every coordinate j to check that $\sum_{i \in [k/\lambda]} d(i, j) \cdot d'(i, j) = \tilde{d}_{sum}(j)$. The cost of each invocation is $O(k/\lambda)$, and we have $O(\lambda)$ such invocations, leading to an overall cost of $O(k)$. In terms of soundness, for each $j \in J$, the probability that the verifier accepts in the underlying multi-sumcheck is at most a constant and so the probability that it accepts for *all* $j \in J$ is $2^{-\lambda}$, as desired.

This concludes the description of the multi-sumcheck protocol. We remark that in prior works, such as [RR21], the multi-sumcheck was the key component and other protocols follows in a straightforward manner. Unfortunately, in our parameter regime this is no longer the case.

To demonstrate this, consider the following related task that arises often in the construction of IOPs. Suppose we are given access to a codeword c and want to check that the first k' bits of c are identically 0. A common approach for doing so is to have the verifier choose at random (or pseudorandomly) a vector $r \in \{0, 1\}^{k'}$ and to run multi-sumcheck on the expression $\sum_{i \in [k']} r_i \cdot c_i = 0$. The point is that if the claim is false (i.e., $c_i \neq 0$ for some $i \in [k']$) then the above sum will still be zero with probability that is inversely proportional to the field size $|\mathbb{F}|$. In all prior works that we are aware of, this sufficed since the goal was to have error probability $1/|\mathbb{F}|$. In contrast, in this work we want to simultaneously work over a field of size $\mathrm{polylog}(\lambda)$ but to have soundness error $2^{-\lambda}$. We manage to solve this difficulty by taking an approach that is similar to, but somewhat more complicated than, our approach for handling the multi-sumcheck protocol.

At a high level, we again view c as a tensor codeword. Using the efficient Reed-Solomon encoding, we can transform c into a new codeword c' so that if c was non-zero in even one coordinate in $[k']$, then c' is non-zero in many of its columns. We can then check each and every one of the columns using [RR21], each with a constant error probability, to overall get a $2^{-\lambda}$ error probability.

A Difficulty with Arithmetization. With a toolkit of efficient IOP sub-protocols in hand, it now seems straightforward to use existing ideas from the literature to construct an IOP for NP. Unfortunately, this turns out to be more complicated than expected. To explain the difficulty, let us focus on a specific NP complete problem which is particularly "arithmetization friendly", called R1CS (for Rank 1 Constraint Satisfiability). We view the problem as being parameterized by three (sparse) square matrices A, B and C and a given input x belongs to the language if there exists w such that $Az \star Bz = Cz$, where $z = (x, w)$.

The typical way to arithmetize the problem is for the prover to send encodings of z, $a = Az$, $b = Bz$ and $c = Cz$ and run sub-protocols to check that:

- $a \star b = c$.

- $a = Az$, $b = Bz$ and $c = Cz$.

The first check can be handled using the multi-sumcheck and related techniques and so we do not elaborate on this point. The latter check however turns out to be more complicated.

Let us see the difficulty in trying to employ a standard approach (c.f., [BCR$^+$19]): the verifier chooses a random (or pseudorandom) vector r to reduce checking that $a = Az$ to checking that:

$$\sum_i r_i a_i = \sum_i r_i (Az)_i = \sum_i (r^T A)_i z_i$$

and now the two sides of the equation can be computed by employing the multi-sumcheck protocol. The problem that we encounter is that a single execution of this approach only has a constant soundness error (if we are working over a small field). Our techniques for boosting the soundness that worked for multi-sumcheck seem to fail because the matrix A is arbitrary which precludes attempts at decomposing the problem into smaller sub-problems.

Rather than handling arbitrary matrices A, we restrict our attention to matrices that have a specific structure. In particular, we start by considering matrices of the form $A = I_k \otimes A_0$, where I_k is the $k \times k$ identity matrix and A_0 is a small matrix, say of size $s \times s$. Intuitively, we can think of A as acting on $k \times s$ matrices, such that if $y = A(x)$, then each row of y is obtained from the corresponding row of x by applying A_0. If x and y are column-wise encoded with a fixed linear code C, then we can say the same thing for the encodings $\hat{x}$ and $\hat{y}$ of x and y respectively: each row of $\hat{y}$ is obtained from the corresponding row of $\hat{x}$ by applying A_0.

Now if C has constant relative distance, the verifier given $\hat{x}$ and $\hat{y}$ has a way to succinctly check, with low soundness error, whether $y = Ax$: the verifier samples λ random rows of $\hat{x}$, along with the corresponding rows of $\hat{y}$, and checks that the latter are obtained from the former by applying A_0. If $y \neq Ax$, then the distance of C implies that at least a constant fraction of the rows of $\hat{y}$ are incorrectly generated, and so the verifier will reject with probability $2^{-\Omega(\lambda)}$. Note that the size of this verifier is $O(\lambda \cdot s^2)$, which can be much smaller than the number of entries of A (which is $k^2 \cdot s^2$).

We can push this simple idea quite far. Instead of viewing A as acting on matrices, we view A as acting on degree-d tensors, and instead of column-wise encodings we use tensor codes. This immediately allows us to handle matrices A of the form $A_1 \otimes \cdots \otimes A_d$. We can also handle sum of such matrices, and in fact we give a more general notion of matrices A that are computable by a new notion that we introduce called "succinct tensor circuits" (see Definition 4.2). As noted in Section 1.1, this notion generalizes (modulo some technicalities) the notion of succinct R1CS [BCG$^+$19], see Appendix B for details.

The Projection Problem. We turn to discussing an additional subtle difficulty that we encounter when attempting to compile our IOP into an efficient argument (and which was briefly discussed in Section 1.1). The common approach for compiling an IOP, as proposed by [BCS16] (following [Kil92]) is for the prover to send Merkle hashes of each of its messages and, at the end of the protocol, to send authentication paths corresponding to the verifier's desired queries.

The problem is that projecting the proof string to the desired query locations may, in general, introduce overhead. This naturally raises the following question: can we construct a "multi-plexing" circuit of size

$O(N) + \text{poly}(k, \log N)$ that given a string $x \in \{0,1\}^N$ (in our case the IOP string) and a set $Q \subseteq [N]$ (i.e., the verifier's queries) of size k, outputs x_Q. Note that for our purposes we cannot afford a circuit of size $\Omega(N \cdot k)$ for this task, and even a circuit of size $\Omega(N \cdot \log(N))$ would be too much.

Unfortunately, we do not know how to solve the above task in general with the desired complexity. Imagine though, that we had a promise that the query set Q has a nice structure - namely, that we can partition the input x into N/k blocks and that the first query in Q is to block 1, the second is to block 2 and so forth. In such a case, we could just concatenate k simple multi-plexing circuits each of size $O(N/k)$ to solve the problem and get, overall, a circuit of size $O(N)$.

Unfortunately, typical IOPs have a highly random query pattern. For example, one of the tasks that we often have to do is sample some $O(\lambda)$ coordinates of a given codeword c (from a code with constant relative distance) so that we can guarantee that with probability $1 - 2^{-\lambda}$ at least one of the entries is non-zero.

Our observation, is that the random sampling of $k = O(\lambda)$ points can indeed be replaced in this case by partitioning the codeword into blocks of N/k and choosing just a single point in each block! While this distribution is far from the uniform distribution over $[n]^k$, it is easy to show that it still solves our sampling task with an exponentially small error probability. We extend this approach to all of the verifier's tests to obtain an IOP for which we can indeed efficiently project the IOP messages to the verifier's query set.

3 Preliminaries

3.1 Notation and Conventions

Throughout this paper we use the notion of a Boolean circuit. When we refer to the size of a circuit, we use the usual meaning (the number of gates).

3.2 Probability

We use the following Chernoff bound.

Fact 3.1 (Chernoff). *If $X_1, \ldots, X_n$ are independent $\{0,1\}$-valued random variables and if μ denotes the expected value of $\sum_i X_i$, then for all $0 \le \delta \le 1$,*

$$\Pr \Big[\sum_i X_i \le (1 - \delta) \cdot \mu \Big] \le e^{-\frac{\delta^2 \mu}{2}}.$$

3.3 Interactive Oracle Proofs

We next define the notion of *interactive oracle proof*, due to [BCS16, RRR21]. Our presentation closely follows [RR21].

We restrict our attention to the public-coin setting which means that all of the verifier's messages simply consist of uniformly random coins.

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, consisting of $(x, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$, 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 $x = (x_{\exp}, x_{\text{imp}})$. 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 $x_{\exp}, R_1, \ldots, R_{\ell-1}$, and outputs the set of query locations I. The circuit $\mathcal{V}_2$ then gets as input $x_{\exp}, R_1, \ldots, R_{\ell-1}$, as well as the projection of $(x_{\mathrm{imp}}, M_1, \ldots, M_\ell)$ to the query set I, denoted by $(x_{\mathrm{imp}}, M_1, \ldots, M_\ell)|_I$, 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., $|\mathcal{V}| := |\mathcal{V}_1| + |\mathcal{V}_2|$.

Remark 3.2. *The standard definition of interactive oracle protocols in the literature (see e.g., [BCS16, RRR21]) 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.

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 3.3 (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 $\ell = 1$.

IOP of Proximity. A particular special case of interest is that of IOP*s of proximity* [BCS16, RRR21], or IOPP for short. For a pair language $\mathcal{L} \subseteq \{(x_{\exp}, x_{\mathrm{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ and $x_{\exp} \in \{0,1\}^*$, we use the notation $\mathcal{L}_{x_{\exp}} := \{x_{\mathrm{imp}} : (x_{\exp}, x_{\mathrm{imp}}) \in \mathcal{L}\}$.

Definition 3.4 (Interactive oracle proof of proximity (IOPP)). *An ℓ-round IOP of α-proximity (α-IOPP) with soundness error ε for a pair language $\mathcal{L} \subseteq \{(x_{\exp}, x_{\mathrm{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ is an ℓ-round IOP with soundness error ε for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where $\mathsf{YES} = \mathcal{L}$ and $\mathsf{NO} = \{(x_{\exp}, y) : y \text{ is } \alpha\text{-far from } \mathcal{L}_{x_{\exp}}\}$.*

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 $\ell = 1$.

Imported Theorem 1 ([RR21, Lemma 6.4 + Proposition 4.7]). *Let $C : \{0,1\}^n \to \{0,1\}$ be a size $S = S(n)$ Boolean circuit. There exists a δ-IOPP for the set $\{x \in \{0,1\}^n : C(x) = 1\}$ with soundness error $1/2$ and the following efficiency parameters:*

- ***Rounds:*** *$O(\log S)$*

- **Prover Size:** $O(S)$

- **Verifier Size:** $O(S) + \mathrm{poly}(\log(S), 1/\delta)$

- **Query Complexity:** $O(1/\delta)$.

3.4 Constructible Finite Fields

We will assume that all fields that we work with are *constructible.*

Definition 3.5. *A field ensemble* $\mathbb{F} = (\mathbb{F}_n)_{n \in \mathbb{N}}$ *is* constructible *if field elements can be represented using* $O(\log(|\mathbb{F}_n|))$ *bits and field operations (i.e., addition, subtraction, multiplication, inversion and sampling random elements) can be performed using a Boolean circuit of size* $\mathrm{polylog}(|\mathbb{F}_n|)$.

Fact 3.6 (See, e.g., [Sho88]). *For every* $S = S(n) \geq 1$, *there exists a constructible field ensemble* $(\mathbb{F}_n)_{n \in \mathbb{N}}$ *where each* $\mathbb{F}_n$ *has characteristic* 2 *and size* $\Theta(S(n))$.

3.5 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 $x \in \mathbb{F}^k$.

3.5.1 Multiplication Codes

Definition 3.7. *The* Hadamard *product of vectors* $x, y \in \mathbb{F}^n$, *denoted* $x \star y$, *is the vector* $z \in \mathbb{F}^n$ *whose* i^{th} *component is* $z_i = x_i \cdot y_i$.

If X *and* Y *are subsets of* $\mathbb{F}^n$, *we write* $X \star Y$ *to denote the set* $\{x \star y : x \in X, y \in Y\}$, *and for integer* $t \geq 0$ *we define*

$$X^{\star t} = \begin{cases} \{1^n\} & \text{if } t = 0 \\ X \star X^{\star(t-1)} & \text{otherwise.} \end{cases}$$

3.5.2 Tensor Codes

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

Definition 3.8 (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 of the* n *resulting rows 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 $C \otimes C'$.

Definition 3.9. *We say that a function $f : \{0,1\}^n \to \{0,1\}^m$ is* locally computable *by a size S circuit if there is a size-S circuit that takes as input $x \in \{0,1\}^n$ and $i \in [m]$ and outputs $f(x)_i$.*

Claim 3.10. *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 Boolean circuits of sizes S, S' respectively, then $C \otimes C'$ can be encoded by a Boolean circuit of size $n' \cdot S + n \cdot S'$.*

2. *If C and C' are locally computable by size S_0 and S'_0 circuits, respectively, then:*

   (a) *There is a size $k' \cdot S_0 + S'_0$ circuit whose input is a message $m \in \mathbb{F}^k \otimes \mathbb{F}^{k'}$ and an index $i \in [n] \times [n']$ and whose output is $(C \otimes C')(m)_i$.*

   (b) *There is a size $S_0 + S'_0$ circuit that on input $m \in \mathbb{F}^k$, $m' \in \mathbb{F}^{k'}$, and $i \in [n] \times [n']$, outputs $(C \otimes C')(m \otimes m')_i$.*

For a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$, let $C^{\otimes 0} := \mathbb{F} \to \mathbb{F}$ denote the identity function (i.e. the function computed by the 1×1 matrix $[1]$), and let and $C^{\otimes t}$ denote $C \otimes C^{\otimes(t-1)}$ for any $t \geq 1$.

Finally, applying iteratively the above Claim 3.10, gives the following.

Claim 3.11. *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 each coordinate of C can be computed in time T_0, then:*

   (a) *For every $t \in \mathbb{Z}^+$, there is a size $O(k^{t-1} \cdot T_0)$ circuit that takes as input $m \in (\mathbb{F}^k)^{\otimes t}$ and $i \in [n]^t$ and outputs $C^{\otimes t}(m)_i$.*

   (b) *For every $t \in \mathbb{Z}^+$, there is a size $O(t \cdot T_0)$ circuit that takes as input $m_1, \ldots, m_t \in \mathbb{F}^k$ and $i \in [n]^t$ and outputs $C^{\otimes t}(m_1 \otimes \cdots \otimes m_t)_i$.*

4 Main Results

Our results are most naturally stated in terms of (a slight modification to the notion of) Rank-1 Constraint Satisfaction (R1CS) relations.

Definition 4.1 (R1CS)**.** *A Rank-1 Constraint Satisfaction (*R1CS*) relation over a field $\mathbb{F}$ is parameterized by matrices $A, B, C \in \mathbb{F}^{M \times N}$ and $X \in \mathbb{F}^{n \times N}$, and is defined as the set*

$$\mathcal{R}_{\mathsf{R1CS}}^{A,B,C,X} \overset{\mathsf{def}}{=} \left\{ (x,z) \in \mathbb{F}^n \times \mathbb{F}^N : X \cdot z = x \ \wedge \ (A \cdot z) \star (B \cdot z) = C \cdot z \right\},$$

where $\star$ denotes pointwise multiplication.

We define $\mathcal{L}_{\mathsf{R1CS}}^{A,B,C,X}$ as the set

$$\left\{ x \in \mathbb{F}^n : \exists z \in \mathbb{F}^N \ s.t. \ (x,z) \in \mathcal{R}_{\mathsf{R1CS}} \right\}.$$

We remark that the standard definition of R1CS require x to be a prefix of z. Our definition can simulate the former by setting X to be the matrix that projects its length-N input to the first n coordinates.

We will focus on R1CS relations for which the matrices A, B, C, X defining the R1CS relation can be succinctly expressed as a "tensor circuit".

Definition 4.2 (Tensor Circuits)**.** *We define a* tensor circuit *over $\mathbb{F}$ to be a triple $C = \left((V_i)_{i \in [N]}, (L_i)_{i \in [g+1]}, (\varphi_i)_{i \in [g]} \right)$, where each V_i is a vector space over $\mathbb{F}$, each L_i is a subset of $[N]$, and for every $j \in [g]$, φ_j is a linear function mapping $\bigotimes_{i \in L_j \setminus L_{j+1}} V_i \to \bigotimes_{i \in L_{j+1} \setminus L_j} V_i$.*

We associate C with a function $f_g \circ \cdots \circ f_1$, where f_i represents a function corresponding to the ith gate φ_i, and is defined as

$$f_i : \bigotimes_{j \in L_i} V_j \to \bigotimes_{j \in L_{i+1}} V_j$$

$$f_i = \varphi_i \otimes \bigotimes_{j \in L_i \cap L_{i+1}} \mathrm{Id}_{V_j} . \tag{1}$$

Remark 4.3. *In Definition 4.2, we take all tensor products to be indexed by an unordered set L. For example, each L_i is an unordered set and yet we consider the tensor product $\bigotimes_{j \in L_i} V_i$. This reflects the fact that many aspects of the "ordering" of wires in (a layer of) a circuit are arbitrary and only serve to complicate notation. For example, when applying a gate φ to the ith and j^{th} wires of a layer, it shouldn't matter whether $i = 1$ and $j = 2$, or if $i = 3$ and $j = 5$. However, if we force the usual ordered semantics, then only the former has the clean notation $\varphi \otimes \mathrm{Id}$. What we are doing with the above notation is implicitly specifying which wires φ is applied to via the domain of φ.*

Definition 4.4 (Tensor Circuit Parameters). *If $C = \big((V_i)_{i \in [N]}, (L_i)_{i \in [g+1]}, (\varphi_i)_{i \in [g]}\big)$ is a tensor circuit, we define the following important parameters of C:*

- ***Width:*** $\max_{i \in [g+1]} \prod_{j \in L_i} \dim(V_j)$.

- ***Degree:*** $\max |L_i|$

- ***Number of Gates:*** g

- ***Total Gate Size:*** *The sum of the circuit complexities of φ_i.*

In terms of tensor circuits, our first main result is the following IOP.

Theorem 4.5 (Main IOP Construction). *Let $\mathcal{R} = \mathcal{R}_{\mathsf{R1CS}}^{A,B,C,X}$ be an R1CS relation, where A, B, and C are $M \times N$ matrices and X is an $n \times N$ matrix.*

Suppose that for some constant-dimensional integer vectors $\mathbf{k}^{(x)}, \mathbf{k}^{(y)}, \mathbf{k}^{(z)}$ (possibly of different dimensions), there are isomorphisms $\mathbb{F}_2^N \cong \bigotimes_i \mathbb{F}^{k_i^{(z)}}$, and $\mathbb{F}_2^M \cong \bigotimes_i \mathbb{F}^{k_i^{(y)}}$, and $\mathbb{F}_2^n \cong \bigotimes_i \mathbb{F}^{k_i^{(x)}}$, such that when A, B, C, and X are viewed as maps between the corresponding tensor spaces, they each are computable by a constant-degree tensor circuit over $\mathbb{F}_2$ with g gates, width W, and total gate size S.

Then for all $\varepsilon > 0$ there is an IOP for $\mathcal{R}$ with soundness error $2^{-\lambda}$ and the following efficiency parameters:

- ***Number of rounds:*** $O(\log N)$.

- ***Prover size:*** $W \cdot g \cdot \mathrm{polylog}(\lambda)$.

- ***Verifier size:*** $O(\lambda \cdot S) + \mathrm{poly}(\lambda, M^\varepsilon, \log n)$, where the polynomial poly is independent of ε, and the verifier is given oracle access to a specific encoding of its input that is computable in time $n \cdot \mathrm{polylog}(\lambda)$.

- ***Queries:*** *The verifier makes $O(\lambda \cdot S)$ queries to the messages sent by the prover, and $O(\lambda)$ queries to the encoding of its input.*

We will use the IOP of Theorem 4.5 to construct an efficient argument-system. As discussed in Section 2 this step is non-trivial.

Theorem 4.6 (Main Interactive Argument Construction). *Assume that there exists a sub-exponentially hard collision-resistant hash function computable by linear size circuits mapping λ bits inputs to $\lambda/2$ bit outputs.*

Let $\mathcal{R} = \mathcal{R}_{\mathsf{R1CS}}^{A,B,C,X}$ be an R1CS relation, where A, B, and C are $M \times N$ matrices and X is an $n \times N$ matrix.

Suppose that for some constant-dimensional integer vectors $\mathbf{k}^{(x)}, \mathbf{k}^{(y)}, \mathbf{k}^{(z)}$ (possibly of different dimensions), there are isomorphisms $\mathbb{F}_2^N \cong \bigotimes_i \mathbb{F}^{k_i^{(z)}}$, and $\mathbb{F}_2^M \cong \bigotimes_i \mathbb{F}^{k_i^{(y)}}$, and $\mathbb{F}_2^n \cong \bigotimes_i \mathbb{F}^{k_i^{(x)}}$, such that when A,

B, C, and X are viewed as maps between the corresponding tensor spaces, they each are computable by a constant-degree tensor circuit over $\mathbb{F}_2$ with g gates, width W, and total gate size S.

Then, for every $\varepsilon > 0$, there is an interactive argument for $\mathcal{R}$ with soundness error $2^{-\lambda}$ and the following efficiency parameters:

- ***Number of rounds:*** $O(\log N)$.

- ***Prover size:*** $W \cdot g \cdot \mathrm{polylog}(\lambda) + \mathrm{poly}(S, \lambda) \cdot (N^\varepsilon + M^\varepsilon)$.

- ***Communication complexity:*** $\mathrm{poly}(\lambda, \log(W \cdot g))$.

- ***Verifier size:*** $\mathrm{poly}(\lambda) \cdot \tilde{O}(S + M^\varepsilon)$, *where the verifier is given oracle access to a specific encoding of its input that is computable by a circuit of size $n \cdot \mathrm{polylog}(\lambda)$.*

Finally, we remark that the interactive argument of Theorem 4.6 can be made *zero-knowledge* using the standard transformation of Ben-Or *et al.* [BGG+88]. The argument can can further be *heuristically* compiled to a non-interactive argument using the Fiat-Shamir transform.

Organization. The technical heart of the proofs are in Sections 5 and 6 and Appendix B. Using these tools Theorems 4.5 and 4.6 are eventually proved in Section 7.

5   IOP Tools

5.1   Projectability

Projection is a map, parameterized by a tuple $(i_1, \ldots, i_k) \in [n]^*$, that takes any string $x \in \{0,1\}^n$ to the subsequence $(x_{i_1}, \ldots, x_{i_k}) \in \{0,1\}^k$. The latter string is called the

6   Basic IOPs

In this section we build the key sub-protocols that will be used to construct our main IOP. We start, in Section 6.1 with an IOP for checking a local multiplicative relation between codewords (aka the Hadamard check) and then proceed to Section 6.2 in which we give an IOP for checking linear relations.

6.1   Hadamard Check

Definition 6.1. *If C is a code, we define Had^C to be the promise problem where "yes" instances are triples $\big(C(x), C(y), C(x \star y)\big)$, and "no" instances are all other triples of codewords of C.*

We first construct an IOP for the Had with constant soundness. The result basically follows from [RR21].

Proposition 6.2 (Constant Soundness Hadamard Check)**.** *Let $\mathbb{F}$ be a constructible field extension of $\mathrm{GF}(2)$. Let $D : \mathbb{F}^k \to \mathbb{F}^n$ be a size-S encodable systematic linear code with constant rate and constant relative distance, and define $C = D^{\otimes t}$ for some constant integer t.*
There is an IOP for Had^C with constant soundness error and the following efficiency parameters:

- ***Rounds:*** $O(\log S)$

- ***Prover size:*** $S \cdot n^{t-1} \cdot \mathrm{polylog}(|\mathbb{F}|)$

- ***Prover communication:*** $O(n^t)$ *elements of $\mathbb{F}$.*

- ***Verifier size:*** $\mathrm{poly}(S, \log(|\mathbb{F}|))$ *for some polynomial* poly *that is independent of t.*

- **Queries:** *The verifier makes a constant number of queries to its input, and for each prover message m the verifier either reads all of m or makes a constant number of queries to symbols in m. Here we view both the input and the prover messages as strings with alphabet $\mathbb{F}$.*

Proof. The proof utilizes a code from [RR21] which supports an efficient "multi-sumcheck" protocol with constant soundness error. We use a simple extension of their construction to a larger alphabet $\mathbb{F}$. This extension incurs additional polylog($|\mathbb{F}|$) factors in computational efficiency for both the prover and verifier, and an $O(\log(|\mathbb{F}|))$ factor in communication.

In more detail, let $\hat{C} = \hat{D}^{\otimes t}$ (where $\hat{D} : \mathbb{F}^k \to \mathbb{F}^{\hat{n}}$ is linear-size encodable) be a code that supports a "multi-sumcheck" protocol with constant soundness error (c.f., [RR21, Lemma 4.8, Proposition 4.7]).

We first describe an encoded-message IOP. Given inputs $c_x = C(x), c_y = C(y), c_z = C(z)$:

1. The prover sends re-encodings $\hat{c}_x = \hat{C}(x), \hat{c}_y = \hat{C}(y), \hat{c}_z = \hat{C}(z)$.

2. The prover and verifier invoke the consistency checking IOP of [RR21, Lemma 5.1], which has constant soundness error, on the claims

$$\left\{ \hat{c}(i) = c(i) \text{ for all } i \in [k]^t \right\}_{(\hat{c},c) \in \{(c_x,\hat{c}_x),(c_y,\hat{c}_y),(c_z,\hat{c}_z)\}}.$$

3. The verifier samples $r_1, \ldots, r_t \in \mathbb{F}^k$ and sends $r = r_1 \otimes \cdots \otimes r_t$ to the prover.

4. The prover and verifier invoke the multi-sumcheck protocol on the claims

$$\sum_{i \in [k]^t} \hat{c}_x(i) \cdot \hat{c}_y(i) \cdot \hat{c}_r(i) = \sum_{i \in [k]^t} \hat{c}_z(i) \cdot \hat{c}_r(i),$$

where we define $\hat{c}_r = \hat{C}(r)$.

Here the verifier needs oracle access not only to $\hat{c}_x, \hat{c}_y, \hat{c}_z : [n]^t \to \mathbb{F}$, but also to $\hat{c}_r$. Using the tensor structure of r, the verifier can emulate oracle access to $\hat{c}_r$ in size $O(t \cdot n)$ by Claim 3.11.

The constant soundness of this protocol follows from the constant soundness of the RR21 protocol, along with the fact that with all but $t/|\mathbb{F}|$ probability over the choice of r, we have $\langle v, r \rangle \neq 0$.

We compile this encoded-message IOP to a standard IOP using [RR21, Proposition 4.7]. $\square$

Our Hadamard check with soundness error $2^{-\lambda}$ overhead only works for a more restricted family of codes, which we fortunately are able to construct with polylog(λ) overhead in the encoding size.

Definition 6.3 ((λ, t)-Hadamard-friendly Codes). *We say that a tensor code C over a field $\mathbb{F}$ is (λ, t)-Hadamard-friendly if it has constant rate and constant relative distance and can be written as $C = E^{\otimes t} \otimes M^{\otimes t}$, where E is any linear code and M is a multiplication code with message space $\mathbb{F}^{\lambda^{1/t}}$ and the property that $M^{\star 2}$ has constant relative distance.*

Lemma 6.4 (Improved Hadamard Check). *Let $t \in \mathbb{Z}^+$ be a constant, and let $\mathbb{F}$ be any constructible field (see Definition 3.5). Let $C \subseteq \mathbb{F}^N$ be a (λ, t)-Hadamard-friendly tensor code.*

There is an IOP for Had^C with soundness error $2^{-\lambda}$ and the following efficiency parameters:

- **Rounds:** $O(\log N)$

- **Communication:** $O(N \cdot \log |\mathbb{F}|)$

- **Prover Size:** $N \cdot \text{polylog}(|\mathbb{F}|)$

- **Verifier Size:** $O\left(\lambda \cdot \left(\lambda + \log N\right) + \text{poly}(N^{1/t}, \log |\mathbb{F}|)\right)$, *where we emphasize that the degree of the polynomial* poly *is independent of t.*

Common Input: Oracle access to triple of codewords $c_x, c_y, c_z \in C = E^{\otimes t} \otimes M^{\otimes t}$

1. The prover computes and sends a codeword $c_z^\star \in E^{\otimes t} \otimes (M^{\star 2})^{\otimes t}$, defined so that $c_z^\star(\alpha, \beta) = c_x(\alpha, \beta) \cdot c_y(\alpha, \beta)$ for all $\alpha \in [k_E]^t$ and $\beta \in [n_M]^t$.

2. The verifier samples $(\alpha_1, \ldots, \alpha_{\kappa_{E \otimes t}}) \leftarrow \mathcal{Q}_{E^{\otimes t}}$ and $(\beta_1, \ldots, \beta_{\kappa_{M \otimes t}}) \leftarrow \mathcal{Q}_{M^{\otimes t}}$.

3. For each $j \in [\kappa_{E^{\otimes t}}]$, and for all $\beta \in [k_M]^t$, the verifier checks that $c_z(\alpha_j, \beta) = c_z^\star(\alpha_j, \beta)$.

4. For each $j \in [\kappa_{M^{\otimes t}}]$:

   (a) Let $\mathbf{u}_j$, $\mathbf{v}_j$, and $\mathbf{w}_j$ respectively denote the vectors $\big(c_x(\alpha, \beta_j)\big)_{\alpha \in [k_E]^t}$, $\big(c_y(\alpha, \beta_j)\big)_{\alpha \in [k_E]^t}$, and $\big(c_z^\star(\alpha, \beta_j)\big)_{\alpha \in [k_E]^t}$. Let $\hat{\mathbf{u}}_j$, $\hat{\mathbf{v}}_j$, and $\hat{\mathbf{w}}_j$ respectively denote $E^{\otimes t}(\mathbf{u}_j) = \big(c_x(\alpha, \beta_j)\big)_{\alpha \in [n_E]^t}$, $E^{\otimes t}(\mathbf{v}_j) = \big(c_y(\alpha, \beta_j)\big)_{\alpha \in [n_E]^t}$, and $E^{\otimes t}(\mathbf{w}_j) = \big(c_z^\star(\alpha, \beta_j)\big)_{\alpha \in [n_E]^t}$.

   (b) The prover and verifier, using oracle access to $\hat{\mathbf{u}}_j$, $\hat{\mathbf{v}}_j$, and $\hat{\mathbf{w}}_j$, invoke the constant soundness error IOPP of Proposition 6.2 on the claim that that $\mathbf{w}_j = \mathbf{u}_j \star \mathbf{v}_j$.

Figure 1: Encoded-message IOP for $\mathsf{Had}^{E^{\otimes t} \otimes M^{\otimes t}}$ with soundness error $2^{-\Omega(\lambda)}$

- **Queries:** $O\big(\lambda^2\big)$. *These queries are prefix-projectable.*

Proof. Let n_E and n_M denote the block lengths of E and M respectively. Let k_E and k_M denote the message lengths of E and M respectively. Let δ_E and δ_M denote the relative distances of E and $M^{\star 2}$ respectively. Let $\mathcal{Q}_{E^{\otimes t}}$ be a prefix-projectable (δ_E^t, λ)-sampler for $[n_E]^t$ and let $\mathcal{Q}_{M^{\otimes t}}$ be a prefix-projectable (δ_M^t, λ)-sampler for $[n_M]^t$ such that:

- $\mathcal{Q}_{E^{\otimes t}}$ and $\mathcal{Q}_{M^{\otimes t}}$ are sampleable by circuits of size $O\big(\lambda \cdot \log n_E\big)$ and $O\big(\lambda \cdot \log n_M\big)$ respectively,

- $\mathcal{Q}_{E^{\otimes t}}$ and $\mathcal{Q}_{M^{\otimes t}}$ both make $O(\lambda)$ queries.

Such $\mathcal{Q}_{E^{\otimes t}}$ and $\mathcal{Q}_{M^{\otimes t}}$ are guaranteed to exist by **??**.

Our construction begins with an *encoded-message* IOP for Had^C, where the prover sends one message that is promised to be a codeword of $E^{\otimes t} \otimes (M^{\star 2})^{\otimes t}$. This protocol is described in Fig. 1.

Completeness Follows immediately from the linearity of $E^{\otimes t}$ and $M^{\otimes t}$ and the completeness of the IOP of Proposition 6.2.

Encoded-Message Soundness We now argue that this IOP has soundness error $2^{-\Omega(\lambda)}$ (this can then be reduced to $2^{-\lambda}$ with constant overhead by repeating in parallel. Let δ_M denote the relative distance of $M^{\star 2}$, and let δ_E denote the relative distance of E.

Case 1: We first claim that if the prover sends $c_z^\star$ that is a valid but incorrectly chosen codeword of $E^{\otimes t} \otimes (M^{\star 2})^{\otimes t}$, then the verifier rejects with all but $2^{-\Omega(\lambda)}$ probability in step 4 (if it has not already rejected in a previous step). Indeed, by the relative distance of $(M^{\star 2})^{\otimes t}$, if $c_z^\star$ is incorrect then for at least δ_M^t fraction of $\beta \in [n_M]^t$, the "column" $\big(c_z^\star(\alpha, \beta)\big)_{\alpha \in [k_E]^t}$ must be incorrect. Then by the sampling properties of $\mathcal{Q}_{M^{\otimes t}}$, it must hold with all but $2^{-\lambda}$ probability over the choice of $\beta_1, \ldots, \beta_{\kappa_{M^{\otimes t}}} \in [n_M]^t$ that for at least λ values of $j \in [\kappa_{M^{\otimes t}}]$, the column $\big(c_z^\star(\alpha, \beta_j)\big)_{\alpha \in [k_E]^t}$ is incorrect (that is, for some $\alpha \in [k_E]^t$, $c_z^\star(\alpha, \beta_j) \neq c_x(\alpha, \beta_j) \cdot c_y(\alpha, \beta_j)$). Thus with all but $2^{-\Omega(\lambda)}$ probability, at least one of the invocations of the IOPP of Proposition 6.2 will cause the verifier to reject.

Case 2: Now suppose that $c_z^\star$ disagrees with c_z on some $(\alpha, \beta) \in [k_E]^t \times [k_M]^t$. We then claim that the verifier will reject with all but $2^{-\Omega(\lambda)}$ probability in Step 3. Indeed, in this case the relative distance of $E^{\otimes t}$ implies that for at least δ_E^t fraction of $\alpha \in [n_E]^t$, the "rows" $\big(c_z^\star(\alpha, \beta)\big)_{\beta \in [k_M]^t}$ and $\big(c_z(\alpha, \beta)\big)_{\beta \in [k_M]^t}$

15

differ. By the sampling properties of $\mathcal{Q}_{M^{\otimes t}}$, it must hold with all but $2^{-\lambda}$ probability over the choice of $\alpha_1, \ldots, \alpha_{\kappa_{E^{\otimes t}}} \in [n_E]^t$ that for at least $\lambda \geq 1$ values of $j \in [\kappa_{E^{\otimes t}}]$, the rows $\left(c_z^\star(\alpha_j, \beta)\right)_{\beta \in [k_M]^t}$ and $\left(c_z(\alpha_j, \beta)\right)_{\beta \in [k_M]^t}$ differ, which causes the verifier to reject.

Finally, suppose that neither case 1 nor case 2 holds. That is, $c_z^\star$ is correctly generated and agrees with c_z on all $(\alpha, \beta) \in [k_E]^t \times [k_M]^t$. Then it must be that $z = x \star y$ as desired.

Efficiency

- **Rounds:** The round complexity is dominated by the IOPP of Step 4, which has $O\left(\log(n_E^{t-1} \cdot S_E)\right) = O(\log n_E)$ rounds.

- **Prover Size:**

  1. The computation of $c_z^\star$ in Step 1 can be done by a circuit of size $k_E^t \cdot n_M^t \cdot \mathrm{polylog}(|\mathbb{F}|) + O\left(n_M^t \cdot n_E^{t-1} \cdot S_E\right) = O\left(N \cdot \mathrm{polylog}(|\mathbb{F}|)\right)$.

  2. The projection of the codewords c_x, c_y, c_z, and $c_z^\star$ onto the rows and columns specified by $\{\alpha_i\}$ and $\{\beta_j\}$ can be done by a circuit of size $O\left(N \cdot \log(|\mathbb{F}|)\right)$ by the (prefix-)projectability of $\mathcal{Q}_{E^{\otimes t}}$ and $\mathcal{Q}_{M^{\otimes t}}$.

  3. In Step 4, the prover for the IOP of Proposition 6.2 is implemented by a circuit of size $n_E^t \cdot \mathrm{polylog}(|\mathbb{F}|)$ for each of the $\kappa_{E^{\otimes t}} = O(\lambda)$ instances, making a circuit of size $\lambda \cdot n_E^t \cdot \mathrm{polylog}(|\mathbb{F}|) = N \cdot \mathrm{polylog}(|\mathbb{F}|)$ in total.

  Adding up these circuit sizes gives the stated bound on the prover complexity.

- **Verifier Size:**

  1. $\mathcal{Q}_{E^{\otimes t}}$ and $\mathcal{Q}_{M^{\otimes t}}$ can, by construction, be sampled by circuits of size $O\left(\lambda \cdot (\log n_E + \log n_M)\right) = O(\log N)$.

  2. In Step 3, the verifier checks equality of $O(\lambda \cdot k_M^t)$ field elements, which requires circuitry of size $O(\lambda \cdot k_M^t \cdot \log |\mathbb{F}|) = O(\lambda^2 \cdot \log |\mathbb{F}|)$.

  3. The verifier for each of the $O(\lambda)$ instances of the IOP invoked in Step 4 has size $\mathrm{poly}(n_E, \log |\mathbb{F}|) = \mathrm{poly}(N^{1/t}, \log |\mathbb{F}|)$ for some polynomial that does not depend on t.

  Adding everything up gives the stated bound on the size of the verifier.

- **Prefix Projectable Queries:** The verifier's queries can be partitioned into two sets of queries — the queries it makes in Step 3 and the queries it makes in Step 4. Recall that the Step 3 queries are made by selecting a prefix-projectable set of "rows" for $E^{\otimes t} \otimes M^{\otimes t}$ and querying c_z and $c_z^\star$ in each entry of these rows. The Step 4 queries are made by choosing a prefix-projectable set of "columns" for $E^{\otimes t} \otimes M^{\otimes t}$, and querying a constant number of entries independently from of each of these column. The prefix-projectability of both types of queries follows from **??**.

Standard Model Soundness Applying **??** gives an IOP in the standard model (where messages are not promised to be valid codewords). $\qquad\square$

6.2 Tensor LinCheck

Definition 6.5 (Lincheck)**.** *If $C : \mathbb{F}^k \to \mathbb{F}^n$ and $C' : \mathbb{F}^{k'} \to \mathbb{F}^{n'}$ are linear codes and $\varphi : \mathbb{F}^k \to \mathbb{F}^{k'}$ is a linear map, we define $\mathsf{Lin}^{C,C',\varphi}$ as the promise problem where:*

- *"Yes" instances consist of pairs $\left(C(x), C'\left(\varphi(x)\right)\right)$,*

- *"No" instances consist of pairs $\left(C(x), C'(y)\right)$ for $y \neq \varphi(x)$.*

We start by describing a basic Lincheck IOP with poor parameters. As a matter of fact, this IOP will not involve the prover or have any interaction. Later, in Lemma 6.8 we will bootstrap this construction and get an efficient IOP.

Lemma 6.6 (Basic Lincheck)**.** *For any constant $t \in \mathbb{Z}^+$, any finite field $\mathbb{F}$, any systematic linear codes $D : \mathbb{F}^{k_D} \to \mathbb{F}^{n_D}$, $D' : \mathbb{F}^{k_{D'}} \to \mathbb{F}^{n_{D'}}$, and $E : \mathbb{F}^{k_E} \to \mathbb{F}^{n_E}$, and any linear $\varphi : \mathbb{F}^{k_D^t} \to \mathbb{F}^{k_{D'}^t}$ that can be implemented by a size-S Boolean circuit, let $\delta_E = \Omega(1)$ denote the relative distance of E, let C denote $D^{\otimes t} \otimes E^{\otimes t}$, and let C' denote $(D')^{\otimes t} \otimes E^{\otimes t}$.*

For $\lambda \in \mathbb{Z}^+$, there is an IOP for $\mathsf{Lin}^{C,C',\varphi \otimes \mathrm{Id}}$ with soundness error $2^{-\lambda}$ and the following efficiency parameters:

- ***Rounds:** 0 (there is no prover involvement)*

- ***Verifier Size:** $O\left(\lambda \cdot (S + \log n_E)\right)$.*

- ***Verifier Randomness:** $O(\lambda \cdot \log n_E)$.*

- ***Query Projectability:** The verifier makes $O\left(\lambda \cdot (k_D^t + k_{D'}^t)\right)$ queries to its input, and these queries are prefix-projectable.*

Common Input: Oracle access to a pair of codewords $c_x \in C, c_y \in C'$.

1. Let $\mathcal{Q}$ be a prefix-projectable (δ_E^t, λ)-sampler for $[n_E]^t$. The existence of such a $\mathcal{Q}$ is guaranteed by **??**. The verifier samples $(i_1, \ldots, i_\kappa) \leftarrow \mathcal{Q}$, where $\kappa = O(\lambda)$.

2. For each $j \in [\kappa]$, let $\mathbf{u}_j$ denote the vector $\left(c_x(\alpha, i_j)\right)_{\alpha \in [k_D^t]}$, let $\mathbf{v}_j$ denote the vector $\left(c_y(\alpha, i_j)\right)_{\alpha \in [k_{D'}^t]}$.

   The verifier checks that $\mathbf{v}_j = \varphi(\mathbf{u}_j)$.

Figure 2: IOP for $\mathsf{Lin}^{C,C',\varphi \otimes \mathrm{Id}}$

Proof. The verifier is described in Fig. 2. The efficiency parameters follow immediately. Query projectability follows from **??**.

It remains to establish that the IOP is complete and sound.

Completeness Suppose we are given $c_x = C(x)$ and $c_y = C'\left((\varphi \otimes \mathrm{Id})(y)\right)$. Let $\mathbf{x}_j = \left(x(i, j)\right)_{i \in [k_D^t]}$ denote the jth "column" of x for $j \in [k_E]$, and similarly let $\mathbf{y}_j$ denote the jth column of y. We are promised that $\mathbf{y}_j = \varphi(\mathbf{x}_j)$ for every $j \in [k_E]$.

Now consider $\hat{y} = (\mathrm{Id} \otimes E^{\otimes t})(y)$ and $\hat{x} = (\mathrm{Id} \otimes E^{\otimes t})(x)$, both of which we can view as $k_D^t \times n_E^t$ matrices. Let $\hat{\mathbf{x}}_j$ and $\hat{\mathbf{y}}_j$ denote the jth columns of $\hat{x}$ and $\hat{y}$ respectively.

By the linearity of $E^{\otimes t}$, it holds for every $j \in [n_E]^t$ that there exist coefficients $\left\{\beta_{i,j}\right\}_{i \in [k_E]^t}$ such that

$$\hat{\mathbf{x}}_j = \sum_{i \in [k_E]^t} \beta_{i,j} \cdot \hat{\mathbf{x}}_i$$

and

$$\hat{\mathbf{y}}_j = \sum_{i \in [k_E]^t} \beta_{i,j} \cdot \hat{\mathbf{y}}_i.$$

17

Thus

$$\hat{\mathbf{y}}_j = \sum_{i \in [k_E]^t} \beta_{i,j} \cdot \hat{\mathbf{y}}_i$$

$$= \sum_{i \in [k_E]^t} \beta_{i,j} \cdot \mathbf{y}_i$$

$$= \sum_{i \in [k_E]^t} \beta_{i,j} \cdot \varphi(\mathbf{x}_i)$$

$$= \varphi\Big(\sum_{i \in [k_E]^t} \beta_{i,j} \cdot \mathbf{x}_i \Big)$$

$$= \varphi(\hat{\mathbf{x}}_i).$$

Since $c_x = (D^{\otimes t} \otimes \mathrm{Id})(\hat{x})$ and $c_y = (D^{\otimes t} \otimes \mathrm{Id})(\hat{y})$, completeness follows by the systematicity of $D^{\otimes t}$.

Soundness Suppose that $c_x = C(x)$ and $\tilde{c}_y = C(\tilde{y})$ are inputs for which $\tilde{y} \neq (\varphi \otimes \mathrm{Id})(x)$. Let c_y denote what $\tilde{c}_y$ is "supposed" to be, i.e. $c_y = C\big((\varphi \otimes \mathrm{Id})(x)\big)$. Since E has constant relative distance δ_E and $\tilde{c}_y \neq c_y$, at least a δ_E fraction of the columns of $\tilde{c}_y$ must differ from c_y. Since D is systematic, these columns are determined by their first k_D entries. Finally, the fact that $\mathcal{Q}$ is a (δ_E, λ)-sampler means that at least λ of the generated claims are false. Thus the verifier rejects with all but $2^{-\Omega(\lambda)}$ probability.

Prefix-Projectability By **??**, we have that $(i_1, \ldots, i_\kappa)$ is prefix-projectable. Since $i_1, \ldots, i_\kappa$ are then used as indexes of rows to be queried in their entirety (or at least in their intersection with the jth column for all j in $[k_D]^t$ or $[k_{D'}]^t$), the prefix-projectability of all of the verifier's input queries follows from **??**. □

Lemma 6.6 admits a generalization to succinct tensor circuits, which is Lemma 6.8 below.

Definition 6.7. *For an integer $t \in \mathbb{Z}^+$ and a vector $\mathbf{k} = (k_1, \ldots, k_d) \in \mathbb{Z}^d$, say C is an $(\mathbb{F}, \mathbf{k}, t)$-tensor code if $C = D^{(1)} \otimes \cdots \otimes D^{(d)}$, where each $D^{(i)}$ has message space $\mathbb{F}^{k_i}$, and each $D^{(i)}$ is also a t-dimensional tensor code $D^{(i)} = (E^{(i)})^{\otimes t}$.*

Lemma 6.8. *Let $\mathbb{F}$ be any field, and for any $\mathbf{k} = (k_1, \ldots, k_d)$, let $\mathcal{M}_{\mathbf{k}}$ denote $\mathbb{F}^{k_1} \otimes \cdots \otimes \mathbb{F}^{k_d}$.*

For any constant $t \in \mathbb{Z}^+$, any $\mathbf{k}$, $\mathbf{k}'$, any $(\mathbb{F}, \mathbf{k}, t)$-tensor code C, and any $(\mathbb{F}, \mathbf{k}', t)$-tensor code C', and any linear function $\phi : \mathcal{M}_{\mathbf{k}} \to \mathcal{M}_{\mathbf{k}'}$ that is computable by a tensor circuit with width W, constant degree, and g gates with total size S, there exists an encoded-message IOP for $\mathsf{Lin}^{C,C',\phi}$ with soundness error $2^{-\lambda}$ and the following efficiency parameters:

- **Rounds:** 1 *(consisting of g encoded messages)*

- **Communication:** $O(g \cdot W)$ *elements of $\mathbb{F}$*

- **Prover Size:** $O(g \cdot W + S)$.

- **Verifier Size:** $O(\lambda \cdot (S + g \cdot \log W))$

- **Queries:** *The verifier makes $O(\lambda \cdot S)$ queries to its input and to the codewords sent by the prover, and these queries are prefix-projectable.*

- **Encoded Messages:** *For all i, the ith prover message is promised to be a codeword of a tensor code $D_i^{\otimes t}$.*

Proof. We first construct a family of codes $\{C_{\mathbf{k}}\}_{\mathbf{k}}$ where $C_{\mathbf{k}}$ is a $(\mathbf{k}, t)$-tensor code that is encodable by a circuit of size $O(|\mathcal{M}_{\mathbf{k}}|)$ and has constant rate and relative distance. For $\mathbf{k} = (k_1, \ldots, k_d)$, we define the code

$C_{\mathbf{k}}$ as $(D_{k_1^{1/t}})^{\otimes t} \otimes \cdots \otimes (D_{k_d^{1/t}})^{\otimes t}$, where $\{D_k\}_{k \in \mathbb{Z}^+}$ is a family of linear-size encodable codes with constant rate and relative distance (e.g. Spielman codes), and with D_k having message space $\mathbb{F}^k$.

The tensor circuit for ϕ expresses φ as a composition of linear maps $\phi = \phi_g \circ \cdots \circ \phi_1$, where each ϕ_i can be viewed as a tensor product $\varphi_i \otimes \mathrm{Id}$. In our IOP on input $\big(C(x), C'(y)\big)$, the prover computes each "intermediate state" $z^{(i)} \stackrel{\mathsf{def}}{=} (\phi_i \circ \cdots \circ \phi_1)(x)$ (which requires circuitry of size $O(S)$), and sends encodings of these states under a "suitable" tensor code.

More specifically, the ith intermediate state $z^{(i)}$ must lie in a message space $\mathcal{M}_{\mathbf{k}^{(i)}}$ where $\mathbf{k}^{(i)} = (k_1^{(i)}, \ldots, k_d^{(i)})$ satisfies $d = O(1)$ and $\prod_j k_j^{(i)} \leq W$. The prover sends an encoding of $z^{(i)}$ under $C_{\mathbf{k}^{(i)}}$, which takes circuitry of size $O(g \cdot W)$ in total and results in $O(g \cdot W)$ field elements of communication. Adding this to the circuitry requirements for generating $z^{(1)}, \ldots, z^{(g)}$ gives the stated bound on the prover complexity.

The verifier then uses the basic lincheck (Lemma 6.6) to check consistency of every pair $\big(z^{(i)}, z^{(i+1)}\big)$ for i $= 0, \ldots, g$, where we denote $z^{(0)} = x$ and $z^{(g+1)} = y$. If φ_i is computable by a circuit of size S_i, then checking consistency of $\big(z^{(i)}, z^{(i+1)}\big)$ takes $O(\lambda \cdot S_i)$ queries, verifier circuitry of size $O(\lambda \cdot S_i)$, and $O(\log W)$ bits of verifier randomness. Summing this over all i gives the stated bounds on the verifier query complexity, the verifier size, and the verifier randomness. Finally, prefix-projectability follows from the prefix-projectability of the input queries in Lemma 6.6. $\qquad\square$

7 Proof of Main Results

7.1 Efficient IOP: Proof of Theorem 4.5

In this section we prove Theorem 4.5. Actually, as we will need it for our second main result, we prove the following strengthening of Theorem 4.5.

Lemma 7.1. *Theorem 4.5 holds and, moreover, the* IOP *verifier's queries are prefix-projectable.*

Proof. We start by constructing an encoded-message IOP with prefix-projectable verifier queries, and then compile it to a standard IOP using **??**.

The main idea is simply to combine Lemmas 6.4 and 6.8. From there, it is a matter of working out what parameters are achieved.

Claim 7.2. *For every $t \in \mathbb{Z}^+$ and constant-dimensional $\mathbf{k} = (k_1, \ldots, k_d)$, there is a tensor code that is simultaneously a $(\mathbf{k}, t)$-tensor code (Definition 6.7) and λ-Hadamard-friendly (Definition 6.3) .*

Proof. Let $\mathbb{F}$ be a field of order $\lambda^{1/t}$, and for any $k \leq \lambda$, let $M_k : \mathbb{F}^k \to \mathbb{F}^{O(k)}$ denote an $\mathbb{F}$-linear, systematic, multiplication code that is encodable by a circuit of size $k \cdot \mathrm{polylog}(\lambda)$, and such that $M_k^{\star 2}$ has constant relative distance. Reed-Solomon codes provide an example of such a code . For every $k \in \mathbb{Z}^+$, let $L_k : \mathbb{F}^k \to \mathbb{F}^{O(k)}$ denote an $\mathbb{F}$-linear, systematic code with constant rate and constant relative distance that is encodable by a circuit of size $O(k \cdot \log \lambda)$ (e.g. a straight-forward generalization of Spielman's code [Spi96]).

Assume without loss of generality that $k_1 \geq \cdots \geq k_d$. If $k_1 > \lambda$, set the code to be

$$(M_{\lambda^{1/t}})^{\otimes t} \otimes L_{(k_1/\lambda)^{1/t}}^{\otimes t} \otimes L_{k_2} \otimes \cdots \otimes L_{k_d}.$$

If $k_1 \leq \lambda$, recursively let C' be a code that is simultaneously a $\big((k_2, \ldots, k_d), t\big)$-tensor code and (λ/k_1)-splittable. Set our code to be $M_{k_1^{1/t}}^{\otimes t} \otimes C'$. $\qquad\square$

Let $D^{(x)}$ denote a $(\mathbf{k}^{(x)}, t)$-tensor code, let $D^{(z)}$ denote a $(\mathbf{k}^{(z)}, t)$-tensor code, and let $D^{(y)}$ denote a code that is simultaneously a $(\mathbf{k}^{(y)}, t)$-tensor code and λ-splittable.

The prover, given z such that $(A \cdot z) \star (B \cdot z) = (C \cdot z)$ and $X \cdot z = x$, computes and sends the following promised codewords to the verifier:

- $c_z = D^{(z)}(z)$, promised to be a codeword of $D^{(z)}$;

19

- $c_A = D^{(y)}(A \cdot z)$, $c_B = D^{(y)}(B \cdot z)$, and $c_B = D^{(y)}(C \cdot z)$, all promised to be codewords of $D^{(y)}$; and

- $c_x = D^{(x)}(x)$.

Upon receiving codewords $\tilde{c}_z$, $\tilde{c}_A$, $\tilde{c}_B$, $\tilde{c}_C$, and $\tilde{c}_x$, the verifier:

- Uses the generalized lincheck (Lemma 6.8) to check that if z denotes the true value encoded in $\tilde{c}_z$ under $C^{(z)}$, then $\tilde{c}_A$, $\tilde{c}_B$, and $\tilde{c}_C$ respectively encode $A \cdot z$, $B \cdot z$, and $C \cdot z$, and $\tilde{c}_x$ encodes $X \cdot z$.

- Uses the improved Hadamard check (Lemma 6.4) to check that if a, b, and c denote the values encoded in $\tilde{c}_A$, $\tilde{c}_B$, and $\tilde{c}_C$, then $c = a \star b$.

$\square$

7.2 From IOPs to Arguments: Proof of Theorem 4.6

In this section we prove Theorem 4.6. To do so, we show how to compile the IOP of Lemma 7.1 into an efficient argument-system. The compiler is based on Kilian's [Kil92] PCP based construction and its extension to IOPs [BCS16] while using linear-time computable hash functions [AHI$^+$17].

The hash functions are used to derive a succinct commitment scheme with local openings.

Definition 7.3. *A* succinct commitment with local openings *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 set $I \in [n]$ outputs the sequence of values $(m_i)_{i \in I} \in \{0,1\}^{|I|}$ as well as a proof $\pi \in \{0,1\}^{\text{poly}(\log n, \lambda)}$. The algorithm* verify *gets as input* crs*, c, I, $(b_i)_{i \in I}$ 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* verify$(\text{crs}, c, I, (m_i)_{i \in I}, \pi) = $ *accept, where* crs $\leftarrow$ setup(n, λ)*, $c = $* commit$(\text{crs}, m)$ *and $\pi \leftarrow $* reveal$(\text{crs}, m, I)$*.*

- *(Succinctness:) The length of c is $\text{poly}(\lambda, \log n)$ and that of π is $\text{poly}(\lambda, \log n, |I|)$.*

- *(Binding:) For every family of polynomial-sized circuits $\mathcal{A} = (\mathcal{A})_\lambda$, the probability over the choice of* crs $\leftarrow$ setup(n, λ) *that $A_\lambda(\text{crs})$ outputs (c, I, π, π') such that* verify$(\text{crs}, c, I, (m_i)_{i \in I}, \pi_0) = $ *accept and* verify$(\text{crs}, c, I, (m'_i)_{i \in I}, \pi') = $ *accept is negligible.*

The following lemma gives a construction of such a commitment scheme using a Merkle tree.

Lemma 7.4. *Assume that there exists a collision-resistant hash function computable by linear size circuits mapping λ bits inputs to $\lambda/2$ bit outputs*
Then, there exists a succinct commitment with local openings such that the commit algorithm can be implemented by a circuit of size $O(n + \lambda)$, the setup can be implemented by a circuit of size $\text{poly}(\log n, \lambda)$, the $verify$ algorithm can be implemented by a circuit of size $\text{poly}(|I|, \log n, \lambda)$.
For any prefix-projectable set $\mathcal{Q}$, there reveal algorithm, restricted to query sets $I \in \mathcal{Q}$ can be implemented in size $O(n) + \text{poly}(\lambda)$.
Moreover, if the collision-resistant hash is sub-exponentially strong, then the resulting commitment is also sub-exponentially small.

Proof Sketch. The construction is simply a Merkle tree hash, and using authentication paths in order to decommit.

In more detail, we use the linear-time encodable hash function, to commit to the Merkle root of the message by a linear-size circuit. To decommit to a sequence I of location, 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). Projecting to the authentication paths is done by utilizing the prefix-projectability of the query set. $\square$

Next, we use the commitment to compile the IOP into an argument-system (c.f., [BCG+17, BCG20, RR21]).

Lemma 7.5. *Assume the existence of a sub-exponentially strong CRHF computable by a linear-size circuit.*

Suppose that the relation $\mathcal{R}$ has an ℓ-round prefix-projectable IOP with soundness error ε and communication complexity c. Then, $\mathcal{R}$ has an $(\ell + 2)$-round argument-system with soundness error $\varepsilon + 2^{-\lambda}$ and communication complexity $O\big((\ell + \log c) \cdot \mathrm{poly}(\lambda)\big)$.

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 has a first offline step that depends only on the input and can be implemented in size P_V and then the online step can be implemented in size T_V.*

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

Sketch. The proof is a (by now standard) extension of Kilian's [Kil92] protocol to IOPs (instead of PCPs), as proposed by Ben Sasson *et al.* [BCS16].Details follow.

The verifier first emulates the offline phase of the original IOP. Then, the interaction involves two phases:

1. **(Interaction :)** The parties emulate the interactive phase of the IOP, except that in every round, rather than sending the IOP message in the clear, the prover generates and sends a commitment to the message using the succinct commitment with local opening.

2. **(Query and Decision Phase:)** In the final stage of the IOP, the verifier needs to query locations in the transcript. To do so, the verifier sends these locations to the prover, who in turn runs the decommitment procedure.

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

Verifier complexity. The verifier's online computation 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 $O(T_P) + \sum_i O(a_i + \lambda) = O(T_P + \ell \cdot \lambda) = O(T_P)$. $\qquad\square$

Lemma 7.6. *Assume that there exists a sub-exponentially hard collision-resistant hash function computable by linear size circuits mapping λ bits inputs to $\lambda/2$ bit outputs.*

Suppose that the relation $\mathcal{R}$ has an ℓ-round public-coin argument-system with soundness error ε, prover complexity T_P and verifier offline complexity P_V and online complexity T_V. Then, $\mathcal{R}$ also has an $\ell + O(1)$-round with soundness error $\varepsilon + 2^{-\lambda}$, communication complexity $\mathrm{poly}(\lambda, \log(T_V))$, prover complexity $T_P + \mathrm{poly}(T_V)$ and verifier complexity $P_V + \mathrm{poly}(\lambda, \log(T_V))$.

Proof Sketch. Loosely speaking, we simply compose the existing argument-system with Kilian's argument-system, details follow.

The prover emulates the interaction but in every round rather than sending the message in the clear, the prover sends a succinct commitment to the message. At the end of the interaction, the prover uses Kilian's [Kil92] argument-system to prove that it knows decommitments that would make the verifier accept. $\qquad\square$

Theorem 4.6 follows by combining Lemmas 7.1, 7.5 and 7.6 (where we choose ε in Lemma 7.1 to dominate the additive fixed polynomial factors in Lemmas 7.5 and 7.6).

Acknowledgements

References

[AHI+17]   Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan, *Low-complexity cryptographic hash functions*, ITCS 2017 (Christos H. Papadimitriou, ed.), LIPIcs, vol. 67, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 7:1–7:31.

[BC12]   Nir Bitansky and Alessandro Chiesa, *Succinct arguments from multi-prover interactive proofs and their efficiency benefits*, CRYPTO 2012 (Reihaneh Safavi-Naini and Ran Canetti, eds.), Lecture Notes in Computer Science, vol. 7417, Springer, 2012, pp. 255–272.

[BCG+17]   Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen, *Linear-time zero-knowledge proofs for arithmetic circuit satisfiability*, ASIACRYPT 2017 (Tsuyoshi Takagi and Thomas Peyrin, eds.), Lecture Notes in Computer Science, vol. 10626, Springer, 2017, pp. 336–365.

[BCG+19]   Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner, *Linear-size constant-query iops for delegating computation*, TCC (2), Lecture Notes in Computer Science, vol. 11892, Springer, 2019, pp. 494–521.

[BCG20]   Jonathan Bootle, Alessandro Chiesa, and Jens Groth, *Linear-time arguments with sublinear verification from tensor codes*, TCC 2020 (Rafael Pass and Krzysztof Pietrzak, eds.), Lecture Notes in Computer Science, vol. 12551, Springer, 2020, pp. 19–46.

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

[BCR+19]   Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward, *Aurora: Transparent succinct arguments for R1CS*, EUROCRYPT 2019, 2019, pp. 103–128.

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

[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*, CRYPTO 1988 (Shafi Goldwasser, ed.), Lecture Notes in Computer Science, vol. 403, Springer, 1988, pp. 37–56.

[BHR+20]   Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni, *Public-coin zero-knowledge arguments with (almost) minimal time and space overheads*, TCC 2020 (Rafael Pass and Krzysztof Pietrzak, eds.), Lecture Notes in Computer Science, vol. 12551, Springer, 2020, pp. 168–197.

[BHR+21]   _____, *Time- and space-efficient arguments from groups of unknown order*, CRYPTO 2021 (Tal Malkin and Chris Peikert, eds.), Lecture Notes in Computer Science, vol. 12828, Springer, 2021, pp. 123–152.

[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*, CRYPTO 2021, Lecture Notes in Computer Science, vol. 12828, Springer, 2021, pp. 92–122.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard, *Perfectly secure multiparty computation and the computational overhead of cryptography*, EUROCRYPT 2010 (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 445–465.

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

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

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass, *Sparks: Succinct parallelizable arguments of knowledge*, EUROCRYPT 2020 (Anne Canteaut and Yuval Ishai, eds.), Lecture Notes in Computer Science, vol. 12105, Springer, 2020, pp. 707–737.

[FKL+21] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng, *Constant-overhead zero-knowledge for RAM programs*, ePrint (2021), 979.

[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.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput. **18** (1989), no. 1, 186–208.

[GRR18] Tom Gur, Govind Ramnarayan, and Ron D. Rothblum, *Relaxed locally correctable codes*, ITCS 2018, 2018, pp. 27:1–27:11.

[HR18] Justin Holmgren and Ron Rothblum, *Delegating computations with (almost) minimal time and space overhead*, FOCS 2018 (Mikkel Thorup, ed.), IEEE Computer Society, 2018, pp. 124–135.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai, *Cryptography with constant computational overhead*, STOC 2008 (Cynthia Dwork, ed.), ACM, 2008, pp. 433–442.

[IKOS09] _____, *Zero-knowledge proofs from secure multiparty computation*, SIAM J. Comput. **39** (2009), no. 3, 1121–1152.

[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)*, STOC 1992, 1992, pp. 723–732.

[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.

[Mic00] Silvio Micali, *Computationally sound proofs*, SIAM J. Comput. **30** (2000), no. 4, 1253–1298.

[RR21] Noga Ron-Zewi and Ron D. Rothblum, *Proving as fast as computing: Succinct arguments with constant prover overhead*, ePrint (2021), 1673.

[RRR21] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum, *Constant-round interactive proofs for delegating computation*, SIAM J. Comput. **50** (2021), no. 3.

[Sho88] Victor Shoup, *New algorithms for finding irreducible polynomials over finite fields*, FOCS 1988, 1988, pp. 283–290.

[Spi96] Daniel A. Spielman, *Linear-time encodable and decodable error-correcting codes*, IEEE Transactions on Information Theory **42** (1996), no. 6, 1723–1731.

[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** (2015), no. 3, 572–598.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang, *Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits*, SP 2021, IEEE, 2021, pp. 1074–1091.

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song, *Libra: Succinct zero-knowledge proofs with optimal prover computation*, CRYPTO 2019, Lecture Notes in Computer Science, vol. 11694, Springer, 2019, pp. 733–764.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang, *Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field*, ePrint (2021), 76.

[ZWZZ20] Jiaheng Zhang, Weijie Wang, Yinuo Zhang, and Yupeng Zhang, *Doubly efficient interactive proofs for general arithmetic circuits with linear prover time*, ePrint **2020** (2020), 1247.

A Interactive Oracle Proofs

We next define the notion of *interactive oracle proof*, due to [BCS16, RRR21]. Our presentation closely follows [RR21].

We restrict our attention to the public-coin setting which means that all of the verifier's messages simply consist of uniformly random coins.

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, consisting of $(x, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$, 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 $x = (x_{\exp}, x_{\imp})$. 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 $x_{\exp}, R_1, \ldots, R_{\ell-1}$, and outputs the set of query locations I. The circuit $\mathcal{V}_2$ then gets as input $x_{\exp}, R_1, \ldots, R_{\ell-1}$, as well as the projection of $(x_{\imp}, M_1, \ldots, M_\ell)$ to the query set I, denoted by $(x_{\imp}, M_1, \ldots, M_\ell)|_I$, 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., $|\mathcal{V}| := |\mathcal{V}_1| + |\mathcal{V}_2|$.

Remark A.1. *The standard definition of interactive oracle protocols in the literature (see e.g., [BCS16, RRR21]) 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.

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 A.2 (Interactive oracle proof (IOP))**.** *An ℓ-round interactive oracle proof (IOP) with soundness error ε for a promise problem (YES, NO) is an ℓ-round (public-coin) interactive oracle protocol $(\mathcal{P}, \mathcal{V})$ such that:*

- **Completeness:** *If $x \in$ YES, then when $\mathcal{V}$ interacts with $\mathcal{P}$, it accepts with probability 1.*

- **Soundness:** *If $x \in$ 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 $\ell = 1$.

IOP of Proximity. A particular special case of interest is that of IOP*s of proximity* [BCS16, RRR21], or IOPP for short. For a pair language $\mathcal{L} \subseteq \{(x_{\exp}, x_{\mathrm{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ and $x_{\exp} \in \{0,1\}^*$, we use the notation $\mathcal{L}_{x_{\exp}} := \{x_{\mathrm{imp}} \colon (x_{\exp}, x_{\mathrm{imp}}) \in \mathcal{L}\}$.

Definition A.3 (Interactive oracle proof of proximity (IOPP))**.** *An ℓ-round IOP of α-proximity (α-IOPP) with soundness error ε for a pair language $\mathcal{L} \subseteq \{(x_{\exp}, x_{\mathrm{imp}}) \in \{0,1\}^* \times \{0,1\}^*\}$ is an ℓ-round IOP with soundness error ε for the promise problem (YES, NO), where YES $= \mathcal{L}$ and NO $= \{(x_{\exp}, y) : y \text{ is } \alpha\text{-far from } \mathcal{L}_{x_{\exp}}\}$.*

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 $\ell = 1$.

Imported Theorem 2 ([RR21, Lemma 6.4 + Proposition 4.7])**.** *Let $C : \{0,1\}^n \to \{0,1\}$ be a size $S = S(n)$ Boolean circuit. There exists a δ-IOPP for the set $\{x \in \{0,1\}^n : C(x) = 1\}$ with soundness error $1/2$ and the following efficiency parameters:*

- **Rounds:** $O(\log S)$

- **Prover Size:** $O(S)$

- **Verifier Size:** $O(S) + \mathrm{poly}(\log(S), 1/\delta)$

- **Query Complexity:** $O(1/\delta)$.

B Iterated Circuit Evaluation and Batch Circuit Satisfiability

We now apply Theorems 4.5 and 4.6 to obtain succinct arguments for succinctly described versions of R1CS that capture iterated circuit evaluation (Appendix B.2) and batch circuit satisfiability (Appendix B.3). Because Theorems 4.5 and 4.6 require that associated linear functions are computable by nice tensor circuits (Definition 4.2), we begin by establishing some basic properties of tensor circuits.

B.1 Basic Properties of Tensor Circuits

We construct tensor circuits to compute a matrix A by writing A as a sum of tensor products of small matrices. The reason this implies a small tensor circuit for A is the following propositions.

Proposition B.1. *If $\varphi = \varphi_1 \otimes \cdots \otimes \varphi_d : F^N \cong \mathbb{F}^{n_1} \otimes \cdots \otimes \mathbb{F}^{n_d} \to \mathbb{F}^{m_1} \otimes \cdots \otimes \mathbb{F}^{m_d} \cong \mathbb{F}^M$ is a linear function, where each φ_i is implementable by a Boolean circuit of size S_i, then f has a d-gate, degree-d tensor circuit with width $O(N + M)$ and total gate size $\sum_i S_i$.*

Proof. This follows straight-forwardly from the definition of a tensor circuit, and from writing $\varphi = f_1 \circ \cdots \circ f_d$, where $f_i = \mathrm{Id}_{n_1} \otimes \cdots \otimes \mathrm{Id}_{n_{i-1}} \otimes \varphi_i \otimes \mathrm{Id}_{n_{i+1}} \otimes \cdots \otimes \mathrm{Id}_{n_d}$. □

Proposition B.2. *If $C_1, \ldots, C_k : F^N \cong \mathbb{F}^{n_1} \otimes \cdots \otimes \mathbb{F}^{n_d} \to \mathbb{F}^{m_1} \otimes \cdots \otimes \mathbb{F}^{m_d} \cong \mathbb{F}^M$ are each (functions computable by) g-gate, degree-d tensor circuits with width W and total gate size S, then $(C_1 + \cdots + C_k)$ is computable by an $O(k \cdot g)$-gate tensor circuit with width $O(k \cdot W)$, degree $d + 1$, and total gate size $O(k^2 \cdot S)$.*

Proof Sketch. The high-level idea is to implement $C_1 + \cdots + C_k$ as the following sequence of steps. On input x:

1. We first map x to $x \otimes 1^k$, which effectively makes k copies of x. This is the reason for the factor k increase in the width of our tensor circuit.

2. For each $i \in [k]$, we emulate each f_i gate-by-gate on the ith copy of x. That is, if φ denotes a gate of f_i, we emulate φ with a function that on input $z \otimes e_j$ for a basis vector $e_j \in \mathbb{F}^k$, outputs $\varphi(z) \otimes e_j$ if $j = i$, and otherwise outputs $z \otimes e_j$. This increases the size of each gate by a factor of k (because the input space's dimension is larger by a factor of k).

3. We collapse the last dimension (which indexes the copies) by applying a function that, for any basis vector $e_i \in \mathbb{F}^k$ for $i \in [k]$, maps $y \otimes e_i$ to y. By linearity, this step maps $\sum_i y_i \otimes e_i$ to $\sum_i y_i$. □

B.2 Iterated Circuit Satisfiability

For a Boolean circuit $D : \{0,1\}^n \to \{0,1\}^n$ and an integer T, define $D^T(x) = D(x)$ if $T = 1$, and $D\big(D^{T-1}(x)\big)$ if $T > 1$. The problem of checking that $D^T(x) = y$ has a tight reduction to the following form of Succinct R1CS [BCG+19]:

Definition B.3 (Succinct R1CS with Initial State Boundary Constraint). *A Succinct R1CS relation with an initial state boundary constraint is a relation $\mathcal{R}_{\mathsf{R1CS}}^{A,B,C,X}$, where:*

- *(Staircase Matrices) The matrices A, B, and C have the $T \times T$ block structure*

$$A = \begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \\ & & & & 0 \end{pmatrix} \quad B = \begin{pmatrix} B_0 & B_1 & & & \\ & B_0 & B_1 & & \\ & & \ddots & \ddots & \\ & & & B_0 & B_1 \\ & & & & 0 \end{pmatrix} \quad C = \begin{pmatrix} C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \\ & & & & 0 \end{pmatrix}$$

  where $A_0, A_1, B_0, B_1, C_0, C_1$ all are $S \times S$ matrices, and the input matrix X is the matrix that projects onto the first S coordinates of its input.

- *(Initial State Boundary Constraint) X is the $S \times (T \cdot S)$ matrix $X_0 + X_T$, where*

$$X_0 = \begin{pmatrix} 1 & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & 1 & 0 & \cdots & 0 \end{pmatrix}$$

and

$$X_T = \begin{pmatrix} 0 & \cdots & 0 & 1 & & \\ \vdots & & \vdots & & \ddots & \\ 0 & \cdots & 0 & & & 1 \end{pmatrix}.$$

We now show that Theorems 4.5 and 4.6 are applicable to succinct R1CS instances with an initial state boundary constraint. That is, we give a tensor decomposition of $\mathbb{F}^{S \cdot T}$ and $\mathbb{F}^n$, and compatible succinct tensor circuits for multiplying by A, B, C, and X. By Propositions B.1 and B.2, it suffices to prove Propositions B.4 and B.6 below.

Proposition B.4 (Staircase Matrix Tensor Decomposition). *For every constant $\varepsilon > 0$, any $kT \times kT$ matrix of the form*

$$\mathbf{A} = \begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \\ & & & & 0 \end{pmatrix},$$

where A_0 and A_1 are $k \times k$ matrices implementable by boolean circuits of size S, can be written in the form

$$\mathbf{A} = \sum_{i=1}^{O(1/\varepsilon)} \bigotimes_{j=1}^{O(1/\varepsilon)} \varphi_{i,j},$$

where each $\varphi_{i,j}$ is computable by a circuit of size $s_{i,j}$ satisfying $\sum s_{i,j} = O(S + T^\varepsilon)$.

Proof. We start by writing $\mathbf{A}$ as

$$\mathbf{A} = I_T \otimes A_0 + \mathsf{LShift}_T \otimes A_1 - M_T \otimes A_0,$$

where we let I_T denote the $T \times T$ identity matrix, we let LShift_T denote the $T \times T$ "left shift" matrix

$$\mathsf{LShift}_T \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix},$$

and we let M_T denote the $T \times T$ matrix

$$M_T \stackrel{\text{def}}{=} \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Note that $M_{n_1 n_2} = M_{n_1} \otimes M_{n_2}$ for any n_1, n_2, and so we can write $M_T = M_{T^\varepsilon}^{\otimes 1/\varepsilon}$. Similarly $I_T = I_{T^\varepsilon}^{\otimes 1/\varepsilon}$. Finally we give a tensor decomposition for LShift_T by applying the following lemma with $k = T^\varepsilon$ and $r = 1/\varepsilon$.

Lemma B.5. *For any k, view $[k^r]$ as in bijective correspondence with $[k]^r$ by associating $i \in [k^r]$ with $(i_1, \ldots, i_r) \in [k]^r$ such that $\sum_{j=1}^{r}(i_j - 1) \cdot k^{r-j} = i - 1$. Correspondingly view $\mathbb{F}^{k^r}$ as $(\mathbb{F}^k)^{\otimes r}$.*

For every k and every r, LShift_{k^r}, viewed as a map on $(\mathbb{F}^k)^{\otimes r}$ via the above correspondence, can be written in the form

$$\mathsf{LShift}_{k^r} = \sum_{i=1}^{r} \bigotimes_{j=1}^{r} \varphi_{i,j},$$

where each $\varphi_{i,j}$ is implementable by a circuit of size k.

Proof. Denote by W_k the $k \times k$ matrix

$$W_k \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \end{pmatrix}$$

We first claim that

$$\mathsf{LShift}_{k^r} = I_{k^{r-1}} \otimes \mathsf{LShift}_k + \mathsf{LShift}_{k^{r-1}} \otimes W_k.$$
$$= I_k^{\otimes r-1} \otimes \mathsf{LShift}_k + \mathsf{LShift}_{k^{r-1}} \otimes W_k. \tag{2}$$

This amounts to verifying that for every $i \in [k^r]$ with corresponding $(i_1, \ldots, i_r) \in [k]^r$, the action of the right-hand side on a basis vector e_i is correct. That is, if $i = 1$ then the output is 0. Otherwise, if $(i_1', \ldots, i_r')$ corresponds to $i - 1$, then the output should be $e_{i_1'} \otimes \cdots \otimes e_{i_r'}$.

Case 0: If $i = 1$, then $i_1 = \cdots = i_r = 1$. We have that $\mathsf{LShift}_k(e_1) = 0$ and $\mathsf{LShift}_{k^{r-1}}(e_1 \otimes \cdots \otimes e_1) = 0$, so the right-hand side of Eq. (2) evaluates to 0 on $e_1 \otimes \cdots \otimes e_1$ as it should.

Case 1: If $i > 1$ and $i_r > 1$, then $W_k(e_{i_r}) = 0$ by definition. We then have

$$\left(I_k^{\otimes r-1} \otimes \mathsf{LShift}_k + \mathsf{LShift}_{k^{r-1}} \otimes W_k\right)(e_{i_1} \otimes \cdots \otimes e_{i_r}) = \left(I_{k^{r-1}} \otimes \mathsf{LShift}_k\right)(e_{i_1} \otimes \cdots \otimes e_{i_r})$$
$$= e_{i_1} \otimes \cdots \otimes e_{i_{r-1}} \otimes e_{i_r - 1}.$$

Indeed, if $(i_1, \ldots, i_r)$ are the base-k digits of an integer i, most-significant digit first, with $i_r > 0$, then the base-k digits of $i - 1$ are $(i_1, \ldots, i_{r-1}, i_r - 1)$. So Eq. (2) is correct in this case.

Case 2: If $i > 1$ and $i_r = 1$, then $\mathsf{LShift}_k(e_{i_r}) = \mathsf{LShift}_k(e_1)$, which is 0 by definition, and $W_k(e_{i_r}) = W_k(e_1) = e_k$. We then have

$$\left(I_k^{\otimes r-1} \otimes \mathsf{LShift}_k + \mathsf{LShift}_{k^{r-1}} \otimes W_k\right)(e_{i_1+1} \otimes \cdots \otimes e_{i_r+1}) = \left(\mathsf{LShift}_{k^{r-1}} \otimes W_k\right)(e_{i_1} \otimes \cdots \otimes e_{i_r})$$
$$= \mathsf{LShift}_{k^{r-1}}(e_{i_1} \otimes \cdots \otimes e_{i_{r-1}}) \otimes e_k.$$

The correctness of this follows from the fact that for any "prefix" $p \in [k^{r-1}]$ corresponding to the $r - 1$ most significant digits of i, we have that $(k \cdot p + 0) - 1 = k \cdot (p - 1) + k - 1$. $\qquad\square$

The proposition then follows by recursively applying Eq. (2), along with the fact that each of I_k, W_k, and LShift_k is implementable by a circuit of size k. $\qquad\square$

Proposition B.6 (Initial State Boundary Constraint). *The matrix X in Definition B.3 can be written as a tensor product of $1 + \frac{1}{\varepsilon}$ matrices whose total circuit size is $O\left(S + \frac{T^\varepsilon}{\varepsilon}\right)$.*

Proof. Let Π denote the $1 \times T^\varepsilon$ matrix with a 1 in the first entry and 0 elsewhere, and let $\tilde{\Pi}$ denote the $1 \times T^\varepsilon$ matrix with a 1 in the last entry and 0 elsewhere. We then have $X_0 = \Pi^{\otimes(1/\varepsilon)} \otimes I_S$ and $X_T = \tilde{\Pi}^{\otimes(1/\varepsilon)} \otimes I_S$, where I_S is the $S \times S$ identity matrix. $\qquad\square$

B.3 Batch Circuit Satisfiability

Theorems 4.5 and 4.6 are also applicable, with much less work, to "batch circuit satisfiability" problem. For integers $n, m \in \mathbb{Z}^+$ and a Boolean circuit $C : \{0,1\}^{n+m} \to \{0,1\}$, let $\mathcal{L}_C$ denote the set $\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m \text{ s.t. } C(x,w) = 1\}$.

If $\mathcal{L}_C = \mathcal{L}_{\mathsf{R1CS}}^{A_0, B_0, C_0, X_0}$, then with $A = I_T \otimes A_0$, $B = I_T \otimes B_0$, $C = I_T \otimes C_0$, and $X = X_0$, we have $\mathcal{L}_C^T = \mathcal{L}_{\mathsf{R1CS}}^{A,B,C,X}$. This immediately implies the following IOPs and arguments.

Proposition B.7. *For any circuit C, integers $T, \lambda \in \mathbb{Z}^+$, and constant $\varepsilon > 0$, there is an IOP for $\mathcal{L}_C^T$ with soundness error $2^{-\lambda}$ with the following efficiency parameters:*

- ***Rounds:*** $O(\log |C| + \log T)$

- ***Communication:*** $O(T \cdot |C|)$

- ***Prover Size:*** $T \cdot |C| \cdot \mathrm{polylog}(\lambda)$

- ***Verifier Size:*** $(|C| + T^\varepsilon) \cdot \mathrm{poly}(\lambda)$

- ***Queries:*** *The verifier's queries are prefix-projectable.*

C From $C^{\otimes t}$-Encoded to Standard IOPs: Proof of ??

In this section we prove **??**. To do so we first recall, and mildly strengthen, some known local properties of tensor codes.

C.1 Local Testability of Tensor Codes

Throughout this section we will be considering the space $[n]^t$ for integers $n, t \in \mathbb{Z}^+$. For distinct $i, j \in [t]$ and $\alpha : ([t] \backslash \{i,j\}) \to [n]$, we define the (i,j,α)-*plane* as the product set $S_1 \times \cdots \times S_t$, where $S_i = S_j = [n]$ and $S_k = \{\alpha(k)\}$ for all $k \in [t] \backslash \{i,j\}$. A set $S \subseteq [n]^t$ is called an (i,j) *plane* if there exists an α such that it is an (i,j,α) plane, and it is simply referred to as a *plane* if there exist i, j and α, so that it is an (i,j,α) plane.

Theorem C.1 (Local testing of tensor codes, [Vid15, Theorem 3.1], (see also [RR21, Theorem B.3]))**.** *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 $w \in \mathbb{F}^n$, if $\mathrm{dist}_\mathbb{F}(w, C^{\otimes t}) \geq \alpha$, then with probability at least $\alpha \cdot \frac{\delta^{2t}}{t^{10}}$ the projection of w to a random plane is not in C^2.*

Lemma C.2. *Let $t \geq 3$ be a constant and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. Then, there exists a randomized oracle algorithm $\mathcal{A}$ which gets as input parameters $\varepsilon, \lambda > 0$ and oracle access to a string $w \in \mathbb{F}^{n^t}$ satisfying the following properties.*

- **Completeness:** *If w is a codeword of $C^{\otimes t}$, then $\mathcal{A}$ accepts with probability 1.*

- **Soundness:** *If $\mathrm{dist}_\mathbb{F}(w, C^{\otimes t}) \geq \varepsilon$, then $\mathcal{A}$ rejects with probability at least $2^{-\lambda}$.*

- **Running time:** *If C can be encoded in time T, then $\mathcal{A}$ has running time $O(n \cdot T \cdot (1/\delta)^{2t} \cdot (1/\varepsilon) \cdot \lambda)$, its query complexity is $O(n^2 \cdot (1/\delta)^{2t} \cdot (1/\varepsilon) \cdot \lambda)$ and randomness complexity is $O(\log(n) \cdot (1/\delta)^{2t} \cdot (1/\varepsilon) \cdot \lambda)$.*

- **Projectability:** *The queries made by $\mathcal{A}$ are prefix projectable.*

Proof. The tester works as follows:

1. For every distinct $i, j \in [t]$:

    (a) Choose random functions $\alpha_1, \ldots, \alpha_\lambda$ from $([t] \backslash \{i,j\})$ to $[n]$ (i.e. choose λ elements of $[n]^{t-2}$) using the prefix-projectable $\left(\alpha \cdot \frac{\delta^{2t}}{t^{10}}, \lambda\right)$-sampler of **??**, for input length $[n]^{t-2}$.

(b) For every $k \in [\lambda]$, check that the restriction of w to the plane (i, j, α_k) belongs to C^2.

(c) If all tests pass then accept, otherwise reject.

To see that completeness holds, note that if w is a codeword, the projection to any plane is also a codeword and so the test passes (with probability 1).

For soundness, if w is α-far from a codeword, then by Theorem C.1, with probability at least $\alpha \cdot \frac{\delta^{2t}}{t^{10}}$, its projection to a random plane is not a codeword of C^2. In particular, there exists $i, j \in [t]$ so that the projection of w to a random (i, j)-plane is not a codeword with similar probability. Fix such an i, j and consider the corresponding iteration of the tester. By our definition of a sampler, $\mathcal{A}$ will reject in this iteration with all but $2^{-\lambda}$ probability as desired.

The complexity of the tester can be easily verified and so we turn to showing that it is prefix-projectable.

We first show a circuit handling the (i, j)-th iteration. For each $\ell_1, \ell_2 \in [n]$ (corresponding to a point on the plane), we can use the circuit guaranteed by **??** using $\alpha_1, \ldots, \alpha_\lambda$ as the query set applied to the input string $x = S_1 \times \cdots \times S_t$, where $S_i = \{\ell_1\}$, $S_j = \{\ell_2\}$ and $S_k = [n]$, for every $k \notin \{i, j\}$. Concatenating these circuits for all ℓ_1, ℓ_2 gives us the projection of the tester's queries for the (i, j)-iteration (namely indices corresponding to the n^2 points in each of the λ planes that it queries). Concatenating the circuits for all (i, j) gives us the desired circuit. $\qquad\square$

C.2 Relaxed Local Correction of Tensor Codes

We will also use the relaxed local correction of tensor codes, as shown by Gur *et al.* [GRR18] (see also [RR21, Lemma B.5]). We extend their result to handle a large number of queries while preserving a projectability property from the query set.

Lemma C.3. *Let $t \geq 1$ be a constant and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a systematic linear code of relative distance δ. there exists a randomized oracle algorithm $\mathcal{A}$ satisfying the following properties.*

- **Input/Output:** *$\mathcal{A}$ takes as input a coordinate set of coordinates $Q \subseteq [n]^t$ and also gets oracle access to a string $w \in \mathbb{F}^{n^t}$. The algorithm outputs either "accept" or "reject".*

- **Completeness:** *If w is a codeword of $C^{\otimes t}$, then $\mathcal{A}$ accepts with probability 1.*

- **Soundness:** *If there exist $c \in C^{\otimes t}$ and $q \in Q$ such that $\text{dist}_{\mathbb{F}}(w, c) < \delta^t/2$ and $w_q \neq c_q$, then the algorithm rejects with all but $2^{-\lambda}$ probability.*

- ***Projectability:*** *For every suffix-projectable set $\mathcal{Q}$, given as in input $Q \in \mathcal{Q}$, the queries made by $\mathcal{A}$ are prefix-projectable.*

- **Other Efficiency Parameters:** *If C can be encoded in time T, then $\mathcal{A}$ has running time $O(|Q| \cdot \lambda \cdot T \cdot (1/\delta)^t)$. $\mathcal{A}$ queries $O(|Q| \cdot \lambda \cdot n \cdot (1/\delta)^t)$ symbols of w, and has randomness complexity $O(\lambda \cdot \log(n) \cdot (1/\delta)^t)$.*

A first glance the statement of Lemma C.3 may seem a bit unusual since the "correcting" algorithm does not output the actual corrected value, but rather just an accept or reject decision. The point however is that, in combination with local testing, the completeness and soundness properties guarantee that if the algorithm accepts, then the projection of c, which is close to w, to the index set Q is simply equal to w_Q.

We further remark that the main novel feature of Lemma C.3, compared to the prior work, is that we achieve a low soundness error while preserving good projectability.

Proof. We will be considering the space $[n]^t$ for integers $n, t \in \mathbb{Z}^+$. For $i \in [t]$ and $\gamma : ([t] \backslash \{i\}) \to [n]$, the (i, γ)-line is defined as the set of points $\{\gamma(1)\} \times \cdots \times \{\gamma(i-1)\} \times [n] \times \{\gamma(i+1)\} \times \cdots \times \{\gamma(t)\}$.

Given two points $\alpha, \beta \in [n]^t$ and an index i, we define the (i, α, β)-*mashup* as the line (i, γ), where $\gamma(j) = \alpha(j)$ for $j < i$ and $\gamma(j) = \beta(j)$ for $j > i$.

We proceed to describe the tester $\mathcal{A}$. Given as explicit input the set of coordinates $Q \subseteq [n]^t$ and oracle access to the string $w \in \mathbb{F}^{n^t}$, $\mathcal{A}$ operates as follows:

1. Choose at random $P \subseteq [n]^t$ from the $(\delta^t/2, \lambda)$-sampler of **??**.

2. For every $p \in P$:

   (a) For every $q \in Q$ and $i \in [t]$, check if the restriction of w to the (i, q, p)-mashup is a codeword of C.

3. If all of the tests passed then accept, otherwise reject.

Completeness. If $c \in C^{\otimes t}$ then its restriction to any axis parallel line is a codeword of C and so the test passes with probability 1.

Soundness. Suppose $\mathrm{dist}_{\mathbb{F}}(w, c) < \delta^t/2$ for some codeword $c \in C^{\otimes t}$ with $c_q \neq w_q$, for some $q \in Q$. In the following we consider only iterations of the tester that correspond to this specific choice of q and ignore all other $q' \neq q$.

Soundness follows immediately from the following proposition, combined with **??**.

Proposition C.4. *If $p \in [n]^t$ is chosen uniformly at random, then the test of Step 2a fails with at least $\delta^t/2$ probability.*

Proof. Let $p = (p_1, \ldots, p_t) \in [n]^t$ be a random point. For every $i \in [t]$, denote by L_i the (i, q, p)-mashup line (which is queried by the tester).

Recall that we now view q as fixed and consider iterating over Step 2a with all values of $i \in [t]$. Note that the i-th iteration of this loop depends only on $p_{i+1}, \ldots, p_t$.

Claim C.5. *Fix $i \in [t-1]$, and randomness $p_{i+1}, \ldots, p_t$, such that $c_{L_i} \not\equiv w_{L_i}$. Then, either the tester rejects in the i-th iteration, or, with probability at least δ over the choice of $p_i \in [n]$, it holds that $c_{L_{i-1}} \not\equiv w_{L_{i-1}}$.*

Proof. Fix $i \in [t-1]$ and randomness $p_{i+1}, \ldots, p_t$ such that $c_{L_i} \not\equiv w_{L_i}$.

If w_{L_i} is not a codeword of C then the tester rejects in the i-th iteration and we are done. Otherwise, both w_{L_i} and c_{L_i} are codewords of C and are *distinct*. Thus, with probability δ over the choice of p_i, they disagree on the point $L_i(p_i) = (q_1, \ldots, q_{i-1}, p_i, p_{i+1}, \ldots, p_t)$. Assume that this is the case and observe that the line $L_{i-1} = (q_1, \ldots, q_{i-2}, *, p_{i+1}, \ldots, p_t)$ passes through this point. Thus, $c_{L_{i-1}} \not\equiv w_{L_{i-1}}$. $\square$

Claim C.6. $c_{L_t} \not\equiv w_{L_t}$.

Proof. The line L_t, which is the (t, q, p) mashup, passes through the point q. The claim follows from the fact that $c(q) \neq w(q)$. $\square$

Claim C.7. *With probability δ^t over the choice of p, it holds that either the tester rejects or $c(p) \neq w(p)$.*

Proof. By Claims C.5 and C.6, with probability δ^{t-1} we have that either the tester rejects or $c_{L_1} \not\equiv w_{L_1}$. We may further assume w_{L_1} is a codeword of C since otherwise the tester rejects. Thus, with probability δ, we have that c and w disagree on the point $L_1(p_1) = p$. $\square$

The proposition now follows from Claim C.7 by observing that $c(p) \neq w(p)$ with probability $\delta^t/2$. $\square$

Efficiency and Projectability. To show prefix projectability, we need to construct a circuit of size $O(t \cdot n^t \cdot \log(|\mathbb{F}|))$ that given as input a codeword $c \in \mathbb{F}^{n^t}$ and the verifier's query set Q, outputs the projection of c to Q, or to prefixes thereof.

We show a separate circuit of size $O(n^t \cdot \log(|\mathbb{F}|))$ for handling each dimension $i \in [t]$. Thus, we can consider a fixed dimension $i \in [t]$ and need to project to the corresponding query set.

Denote the set of random points selected by the tester by P. The tester's queries then correspond to all of the (i, p, q)-mashup lines for all $q \in Q$ and $p \in P$.

To do so the circuit first generates the projection of c to all of the subspaces $(q_1, \ldots, q_{i-1}, *, \ldots, *)$. Since the set Q is prefix projectable, this can be done using a circuit of size $O(n^i \cdot \log(|\mathbb{F}|))$ with $\min(|Q|, n^i) \cdot \log(|\mathbb{F}|)$ outputs, where we use the fact that the total number of possible projections is n^i.

Next, we project each output subspace (which has dimension $t - i$) to the set of lines $(*, p_{i+1}, \ldots, p_t)$, each of which can be done by a circuit of size $O(n^{t-i})$ (see **??**), and produces the projection to the mashup line (i, q, p). Overall we get a circuit of size $O\big((n^i + \min(|Q|, n^i) \cdot n^{t-i}) \cdot \log(|\mathbb{F}|)\big) = O\big(n^t \cdot \log(|\mathbb{F}|)\big)$, as desired.

The other efficiency parameters follow from the construction.

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ □

C.3   Proof of ??

The prover and verifier emulate the original IOP protocol. We refer to round in which, according to the specification of the protocol, the prover should send $C^{\otimes t}$ encoded messages as "encoded rounds". Let r denote the number of encoded rounds.

In addition to emulating the original protocol, in every encoded round, if the prover sends a message w, the verifier runs the local testing procedure of Lemma C.2 on w with proximity parameter $\delta^t/2$. The verifier also runs the relaxed local correction procedure of Lemma C.3 to check that its query set Q to w has the correct answers.

Completeness follows from the completeness of the original protocols together with the completeness conditions of Lemmas C.2 and C.3.

For soundness, if in some round the prover sends a message that is more than $\delta^t/2$-far from the code then, by Lemma C.2 the verifier rejects with all but $2^{-\lambda}$ probability and we are done. Thus, we may assume that the prover always sends close messages. Assume that one of these messages disagrees with the underlying codeword in one of the verifier's query locations. Then, by the soundness condition of Lemma C.3, the verifier also rejects with all but $2^{-\lambda}$ probablility.

Thus, we may assume that both of these events do not occur. Soundness now follows from $C^{\otimes t}$ encoded-message soundness of the original IOP. Prefix projectability of the final IOP follows from the prefix-projectability of the original IOP as well as the prefix-projectability guaranteed by Lemmas C.2 and C.3.