Linear-Time Arguments with Sublinear Verification from Tensor Codes

Jonathan Bootle
jonathan.bootle@berkeley.edu
UC Berkeley

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Jens Groth
jens@dfinity.org
Dfinity

December 28, 2020

Abstract

Minimizing the computational cost of the prover is a central goal in the area of succinct arguments. In particular, it remains a challenging open problem to construct a succinct argument where the prover runs in linear time and the verifier runs in polylogarithmic time.

We make progress towards this goal by presenting a new linear-time probabilistic proof. For any fixed $\epsilon > 0$, we construct an interactive oracle proof (IOP) that, when used for the satisfiability of an N-gate arithmetic circuit, has a prover that uses $O(N)$ field operations and a verifier that uses $O(N^\epsilon)$ field operations. The sublinear verifier time is achieved in the holographic setting for every circuit (the verifier has oracle access to a linear-size encoding of the circuit that is computable in linear time).

When combined with a linear-time collision-resistant hash function, our IOP immediately leads to an argument system where the prover performs $O(N)$ field operations and hash computations, and the verifier performs $O(N^\epsilon)$ field operations and hash computations (given a short digest of the N-gate circuit).

Keywords: interactive oracle proofs; tensor codes; succinct arguments

Contents

1 Introduction

Succinct arguments are cryptographic proofs for NP in which the number of bits exchanged between the argument prover and the argument verifier is much less than the size of the NP witness (e.g., polylogarithmic in the size of the NP witness). Succinct arguments originate in the seminal works of Kilian [Kil92] and Micali [Mic00], and have now become the subject of intense study from theoreticians and practitioners, with a great deal of effort invested in improving their asymptotic and concrete efficiency.

The main efficiency measures in a succinct argument are communication complexity (the number of bits exchanged between the prover and the verifier), as well as the running time of the prover and the running time of the verifier. Over the last decade there has been much progress in improving the communication complexity and verifier time for succinct arguments whose prover runs in quasilinear time. These advances have, in particular, enabled real-world deployments of succinct arguments as part of security systems where the succinct argument is used to certify correctness of certain medium-size computations (e.g., [Ben+14]).

There are, however, exciting envisioned applications where the succinct argument is used to prove the correctness of large-scale computations (see [OWWB20] and references therein). While a proving time that is quasilinear is arguably asymptotically efficient, the polylogarithmic overheads severely limit the sizes of computations that can be supported in applications, because proving time quickly becomes a bottleneck.

This state of affairs motivates the fundamental problem of constructing *linear-time succinct arguments*: succinct arguments where the prover runs in linear time and, ideally, also where the verifier runs in sublinear (e.g., polylogarithmic) time. In this paper we present new constructions that make progress on this problem.

Challenges. There are different approaches for constructing succinct arguments, yet essentially all of them follow the same high-level pattern: first, *arithmetize* the computation whose correctness is being proved; second, *probabilistically check* the arithmetized problem via the help of cryptography. Typically, the first step alone already costs more than linear time because it involves, in particular, encoding the computation as a polynomial, an operation that can be performed in quasilinear time thanks to the Fast Fourier Transform (FFT) but is not believed to have a linear-time algorithm. This means that many of the algebraic techniques that have proven useful to construct succinct arguments seem inapplicable in the linear-time regime.

Prior work. Few works achieve some form of succinct argument without using FFTs, and none of them resolve the problem of constructing linear-time succinct arguments. We briefly review these works below, and also compare their main features in Figure 1 (alongside the arguments that we construct in this paper).

Several works [BCCGP16; BBBPWM18; WTSTW18; XZZPS19; Set20] forego the use of FFTs by using homomorphic commitments to realize a "cryptographic arithmetization", but in doing so also introduce quasilinear work in the cryptography. In some works the quasilinear costs, due to the cryptography [XZZPS19] or an FFT [ZXZS20], can be isolated to the witness of the non-deterministic computation and thereby achieve linear work if the witness is sufficiently small; but, in general, the witness may be as large as the computation.

While the above works achieve polylogarithmic communication complexity but not linear-time proving, Bootle et al. [BCGGHJ17] achieve linear-time proving with square-root communication complexity (and verification): an argument system for arithmetic circuit satisfiability where, for an N-gate circuit, the prover performs $O(N)$ field operations and hash computations while the verifier performs $O(\sqrt{N})$ field operations and hash computations, with a communication complexity of $O(\sqrt{N})$. Crucially, the hash function is only required to be collision resistant, for which there are linear-time candidates (e.g., assuming the intractability of certain shortest vector problems [AHIKV17]), which leads to a linear-time prover.

Overall, the construction in [BCGGHJ17] remains the only argument system for NP known to date where the prover runs in linear time and where communication complexity is sublinear. Improving on the square-root communication complexity, and ideally also the square-root verifier time, is an open problem.

Linear-time IOPs suffice. The approach used by Bootle et al. [BCGGHJ17] to obtain their linear-time argument system highlights a natural target for improvement, as we now explain. First, they construct an *interactive oracle proof* (IOP) with prover time $\mathtt{tp} = O(N)$, query complexity $\mathtt{q} = O(\sqrt{N})$, and verifier time $\mathtt{tv} = O(\sqrt{N})$. (An IOP is a "multi-round PCP" [BCS16; RRR16], as we will review later on.) Second, they apply the "commit-then-open" paradigm of Kilian [Kil92], by using a collision-resistant hash function to transform the IOP into an argument system where communication complexity is $O(\mathtt{q} \cdot \log N)$. In this latter step, if one can evaluate the hash function in linear time, the resulting argument prover runs in time $O(\mathtt{tp})$ and $O(\mathtt{tv})$. We see here that, given linear-time hash functions, *the problem of constructing linear-time succinct arguments reduces to constructing linear-time IOPs with small query complexity (and verifier time).*

In other words, the target for improvement is the IOP. Our goal in this paper is to construct an IOP with linear-time prover whose query complexity and verifier time improve on the prior art, which would yield an argument system with corresponding improvements. For example, improving query complexity to be polylogarithmic would yield the first linear-time argument with polylogarithmic communication complexity.

We conclude here by noting that the above approach has the additional benefit of being plausibly post-quantum, as the underlying linear-time hash function candidate is based on a lattice problem [AHIKV17].

1.1 Our results

We construct, for any fixed $\epsilon > 0$, an argument system where the prover performs $O(N)$ field operations and hash computations, communication complexity is $O(N^\epsilon)$, and the verifier performs $O(N^\epsilon)$ field operations and hash computations. We achieve this by improving the state of the art in linear-time IOPs (see Figure 2): our main result is a public-coin IOP where, for any fixed $\epsilon > 0$, the prover performs $O(N)$ field operations, query complexity is $O(N^\epsilon)$, and the verifier performs $O(N^\epsilon)$ field operations. These costs are when proving the satisfiability of an N-gate arithmetic circuit defined over any field of size $\Omega(N)$.[1]

In more detail, we focus on constructing protocols for *rank-1 constraint satisfiability* (R1CS), a standard generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices.[2]

Definition 1 (informal)**.** *The R1CS problem asks: given a finite field $\mathbb{F}$, coefficient matrices $A, B, C \in \mathbb{F}^{N \times N}$ each containing at most $M = \Omega(N)$ non-zero entries, and an instance vector x over $\mathbb{F}$, is there a witness vector w over $\mathbb{F}$ such that $z := (x, w) \in \mathbb{F}^N$ and $Az \circ Bz = Cz$? (Here "$\circ$" denotes the entry-wise product.)*

Theorem 1 (informal)**.** *For every positive constant $\epsilon > 0$, there is a public-coin holographic IOP for R1CS, over any field of size $\Omega(M)$, with the following parameters:*

- *round complexity is $O(1/\epsilon + \log M)$;*
- *proof length is $O(M)$ elements in $\mathbb{F}$;*
- *query complexity is $O(M^\epsilon)$;*
- *the prover uses $O(M)$ field operations; and*
- *the verifier uses $O(M^\epsilon)$ field operations, given access to a linear-time encoding of the coefficient matrices.*

Our theorem directly follows from two results of independent interest. First, we construct a proof protocol for R1CS with a linear-time prover, but in an intermediate model that extends the type of queries that the

[1] The sublinear time of the argument verifier is achieved in the preprocessing setting, which means that the verifier receives as input a short digest of the circuit that can be derived by anyone (in linear time). Some form of preprocessing is necessary for sublinear verification because the argument verifier just reading the circuit takes linear time. In turn, preprocessing is enabled by the fact that our IOP is holographic, which means that the IOP verifier has oracle access to a linear-size encoding of the circuit that is computable in linear time. See [CHMMVW20; COS20] for more on how holography leads to preprocessing.

[2] Recall that satisfiability of an N-gate arithmetic circuit is reducible, in linear time, to an R1CS instance where the coefficient matrices are $N \times N$ and have $O(N)$ non-zero entries.

verifier can make in an IOP. Second, we efficiently "implement" this intermediate model via a standard IOP. We summarize each of these two results below. The formal statement of Theorem 1 is in Section 4.

We remark that our result, unlike many other results about efficient probabilistic proofs, holds over *any* field $\mathbb{F}$ that is large enough (linear in M) without requiring any special structure (e.g., smooth subgroups).

(1) IOP with tensor queries for R1CS. We use the notion of a *tensor IOP*, which is an IOP where the verifier can make *tensor queries* to the proof strings sent by the prover, as opposed to just point queries as in a standard IOP. To make a tensor query to one of the received proof strings, the verifier specifies a vector with prescribed tensor structure and receives as answer the inner product of the tensor vector and proof string.

Definition 2 (informal)**.** *A $(\mathbb{F}, k, t)$-tensor IOP modifies the notion of an IOP as follows: (a) the prover message in each round i is a string Π_i in $\mathbb{F}^{\ell_i \cdot k^t}$ for some positive integer ℓ_i; (b) a verifier query may request the value $\langle a_0 \otimes a_1 \otimes \cdots \otimes a_t, \Pi_i \rangle$ for a chosen round i and chosen vectors $a_0 \in \mathbb{F}^{\ell_i}$ and $a_1, \ldots, a_t \in \mathbb{F}^k$.*

The first part to our proof of Theorem 1 is a $(\mathbb{F}, k, t)$-tensor IOP for R1CS with a $O(M)$-time prover, constant query complexity, and a $O(M^{1/t})$-time verifier (who has tensor-query access to the coefficient matrices).

Theorem 2 (informal)**.** *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor IOP for R1CS that supports coefficient matrices in $\mathbb{F}^{N \times N}$ with $N = k^t$ and up to $M = O(N)$ non-zero entries and has the following parameters:*

- *soundness error is $O(M/|\mathbb{F}|)$;*
- *round complexity is $O(\log N)$;*
- *proof length is $O(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(1)$;*
- *the prover uses $O(M)$ field operations; and*
- *the verifier uses $O(M^{1/t})$ field operations, given tensor-query access to the coefficient matrices.*

We sketch the ideas behind this result in two steps: in Section 2.4 we describe a tensor IOP for R1CS achieving all efficiency parameters except that the verifier explicitly reads the coefficient matrices and uses $O(M)$ field operations; then in Section 2.5 we describe how to extend this tensor IOP to the holographic setting, achieving a sublinear verifier time when the verifier is granted tensor-query access to the coefficient matrices. The corresponding technical details are provided in Section 5 and Section 6, respectively. From a technical perspective, our construction builds on tools from several papers, such as linear-time scalar products in [BCGGHJ17], linear-time sumchecks in [Tha13; XZZPS19], and linear-time look-ups in [Set20; GW20].

(2) From tensor queries to point queries. We prove that any tensor IOP can be efficiently implemented as a standard IOP, by way of a subprotocol that "simulates" tensor queries via a collection of point queries.

In more detail, we provide a transformation that receives as input a tensor IOP and any linear code represented via a circuit for its encoding function, and produces as output a point-query IOP that decides the same language as the tensor IOP up to an additional soundness error.

The key efficiency feature of the transformation is that prover complexity is preserved up to the number of tensor queries, the code's rate, and the code's encoding time. In particular, if the prover in the tensor IOP uses a linear number of field operations and the verifier makes a constant number of tensor queries, and the code is linear-time encodable, then the new prover in the standard IOP uses a linear number of field operations. In the following theorem, and throughout the paper, we use "Big O" notation such as $O_a(\cdot)$, which means that the parameter a is treated as a constant.

Theorem 3 (informal)**.** *There is an efficient transformation that takes as input a tensor-query IOP and a linear code, and outputs a point-query IOP that has related complexity parameters, as summarized below.*

- Input IOP: *an* $(\mathbb{F}, k, t)$*-tensor IOP with soundness error* ϵ*, round complexity* rc*, proof length* l*, query complexity* q*, prover arithmetic complexity* tp*, and verifier arithmetic complexity* tv.
- Input code: *a linear code* $\mathcal{C}$ *over* $\mathbb{F}$ *with rate* $\rho = \frac{k}{n}$*, relative distance* $\delta = \frac{d}{n}$*, and encoding time* $\theta(k) \cdot k$.
- Output IOP: *a point-query IOP with soundness error* $O_{\delta,t}(\epsilon) + O(d^t/|\mathbb{F}|)$*, round complexity* $O_t(\mathsf{rc})$*, proof length* $O_{\rho,t}(\mathsf{q} \cdot \mathsf{l})$*, query complexity* $O_t(k \cdot \mathsf{q})$*, prover arithmetic complexity* $\mathsf{tp} + O_{\rho,t}(\mathsf{q} \cdot \mathsf{l}) \cdot \theta(k)$*, and verifier arithmetic complexity* $\mathsf{tv} + O_t(k \cdot \mathsf{q}) \cdot \theta(k)$.

Moreover, the transformation preserves holography up to the multiplicative overhead θ *induced by the encoding function of* $\mathcal{C}$ *and factors that depend on* ρ *and* t.

We stress that the *only* property of the code $\mathcal{C}$ used in the above transformation is that it is linear over $\mathbb{F}$, and in particular the code $\mathcal{C}$ need not be efficiently decodable, satisfy the multiplication property (entry-wise multiplication of codewords is a codeword in a related code), or even be systematic. We believe that developing techniques that work with a wide range of codes will facilitate further IOP research. For example, known linear-time encodable codes meeting the Gilbert–Varshamov bound are not systematic [DI14]; also, efficient zero knowledge (not a goal in this paper) is typically achieved by using non-systematic codes.

We sketch the ideas behind this result in Sections 2.2 and 2.3. The technical details are in Sections 8 to 10. From a technical perspective, our transformation builds on ideas from several papers: the sumcheck protocol for tensor codes in [Mei13]; the ILC-to-IOP compiler in [BCGGHJ17] that works with any linear code; the proximity test for the Reed–Solomon code in [BBHR18]; and the code-switching technique in [RR20] for systematic linear codes.

	preprocess circuit cost	prover cost	verifier cost	communication complexity	plausibly post-quantum
[BCCGP16] & [BBBPWM18]	n/a	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O_\lambda(\log N)$	✗
[WTSTW18]	n/a	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O(D \log W)$ $\mathbb{F}$-ops $O(N^\epsilon + D \log W)$ $\mathbb{G}$-exps	$O_\lambda(N^{1-\epsilon}$ $+D \log W)$	✗
[XZZPS19]	n/a	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O(D \log W)$ $\mathbb{F}$-ops $O(\log W)$ pairings	$O_\lambda(D \log W)$	✗
[Set20]	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O(N)$ $\mathbb{F}$-ops $O(N)$ $\mathbb{G}$-exps	$O(\log^2 N)$ $\mathbb{F}$-ops $O(\log^2 N)$ $\mathbb{G}$-exps	$O_\lambda(\log^2 N)$	✗
[BCGGHJ17]	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O_\lambda(\sqrt{N})$ $\mathbb{F}$-ops $O_\lambda(\sqrt{N})$ hashes	$O_\lambda(\sqrt{N})$	✓
this work	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O_\lambda(N^\epsilon)$ $\mathbb{F}$-ops $O_\lambda(N^\epsilon)$ hashes	$O_\lambda(N^\epsilon)$	✓

Figure 1: Comparison of several sublinear argument systems that do not use FFTs. The stated costs are for the satisfiability of an N-gate arithmetic circuit over a cryptographically-large field $\mathbb{F}$; for argument systems that achieve sublinear verification we also report the cost to preprocess the circuit. We report separate costs for field operations, group operations, and (collision-resistant) hash invocations; ϵ is any positive constant and λ is the security parameter. Provers for arguments in the top part of the table run in superlinear time. Indeed, $O(N)$ exponentiations in $\mathbb{G}$ result in $\omega(N)$ group operations: $O(\log |\mathbb{F}| \cdot N)$ group operations if performed naively, or else $O(\frac{\log |\mathbb{F}|}{\log \log |\mathbb{F}| + \log N} \cdot N)$ if using Pippenger's algorithm [Pip80]. On the other hand, provers in the bottom part of the table run in linear time. Indeed, as observed in [BCGGHJ17], by using the hash functions of [AHIKV17] one can ensure that $O(N)$ hash invocations are equivalent, up to constants, to $O(N)$ operations in $\mathbb{F}$. The argument systems in [WTSTW18; XZZPS19] specifically require the circuit to be arranged in layers; the reported costs are for a circuit with D layers of width W, in which case $N = D \cdot W$; furthermore the term "$O(D \log W)$ $\mathbb{F}$-ops" in the verifier cost assumes that the circuit is sufficiently uniform and, if not, increases to "$O(N)$ $\mathbb{F}$-ops" (i.e., linear in computation size).

point-query IOPs	encode circuit cost	prover cost	verifier cost	query complexity
[BCGGHJ17]	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(\sqrt{N})$ $\mathbb{F}$-ops	$O(\sqrt{N})$
this work	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(N^\epsilon)$ $\mathbb{F}$-ops	$O(N^\epsilon)$

Figure 2: Comparison of known IOPs with a linear-time prover. The parameters are for an N-gate arithmetic circuit defined over a field $\mathbb{F}$ of size $\Omega(N)$; and ϵ is any positive constant. The sublinear verification in both cases is achieved in the holographic setting (the verifier has oracle access to an encoding of the circuit).

2 Techniques

We summarize the main ideas behind our results. We begin by elaborating on our main result, Theorem 1, which is a new protocol within a proof model called *Interactive Oracle Proof* (IOP) [BCS16; RRR16].

Recall that an IOP is a proof model in which a prover and a verifier interact over multiple rounds, and in each round the prover sends a proof message and the verifier replies with a challenge message. The verifier has query access to all received proof messages, in the sense that it can query any of the proof messages at any desired location. The verifier decides to accept or reject depending on its input, its randomness, and answers to its queries. The main information-theoretic efficiency measures in an IOP are proof length (total size of all proof messages) and query complexity (number of read locations across all proof messages), while the main computational efficiency measures are prover running time and verifier running time.

In this paper we study IOPs because they directly lead to corresponding succinct arguments, via cryptography that introduces only *constant* computational overheads (and in particular preserves linear complexity).[3] Namely, following the paradigm of Kilian [Kil92], any IOP can be "compiled" into a corresponding interactive argument by using a collision-resistant hash function. The argument's communication complexity is $O(\mathsf{q} \log \mathsf{l})$, where q and l are the query complexity and the proof length of the IOP.[4] Moreover, with a suitable choice of hash function (e.g., [AHIKV17]), the running times of the argument prover and argument verifier are the same, up to multiplicative constants, as those of the IOP prover and IOP verifier.[5]

The rest of this section summarizes the proof of Theorem 1. We proceed in three steps. First, we describe an intermediate proof model called tensor IOPs; we elaborate on this model in Section 2.1. Second, we devise a transformation that, using an arbitrary linear code, efficiently "implements" any tensor IOP as a point-query (standard) IOP; this is our Theorem 3, and we discuss the transformation in Sections 2.2 and 2.3. Third, we construct a tensor IOP with linear-time prover, constant query complexity, and sublinear-time verifier; this is our Theorem 2, and we discuss this construction in Sections 2.4 and 2.5.

2.1 IOPs with tensor queries

In this work we rely on an intermediate model, informally introduced in Definition 2, called *tensor IOPs*. Below we briefly elaborate on why we introduce this model, and also compare it with other existing models.

Point queries are for efficiency. The verifier in an IOP makes *point queries* to proof messages received from the prover: the verifier may specify a round i and a location j and then receives as answer $\Pi_i[j]$ (the j-th value of the proof message Π_i sent in round i). Our main result (Theorem 1) is about point-query (standard) IOPs because, as we explained, they lead to succinct arguments via constant computational overheads.

Beyond point queries. Researchers have studied variants of the IOP model where the verifier makes other types of queries. For example, Boneh et al. [BBCGI19] study *linear IOPs*, where the verifier may specify a round i and a vector q and then receives as answer the linear combination $\langle q, \Pi_i \rangle$, over a field $\mathbb{F}$. These $\mathbb{F}$-linear queries are a "richer" class because linear combinations can, in particular, select out chosen locations.

[3] We stress that this is a non-trivial property, in the sense that other approaches to construct succinct arguments introduce *super-constant* multiplicative overheads. For example, the transformation from algebraic proofs to succinct arguments in [CHMMVW20] introduces a linear number of exponentiations (which translates to a super-linear number of group operations). These approaches seem unlikely to lead to linear-time succinct arguments, and hence we focus on IOP-based succinct arguments.

[4] The "big O" notation here hides a dependence on the output size of the collision-resistant hash function.

[5] We remark that the more restricted proof model of Probabilistically Checkable Proofs (PCPs) also directly leads to a succinct argument with only constant computational overheads, however the problem of designing linear-time PCPs, with *any* non-trivial query complexity, seems far beyond current techniques.

From the perspective of this paper, variants such as linear IOPs offer an opportunity to reduce our goal (a certain point-query IOP) into two sub-problems. First, design an efficient IOP with a richer class of queries. Second, devise a way to efficiently "implement" the rich class of queries via only point queries. The former becomes easier as the class of queries becomes richer, while the latter becomes harder. Thus, the class of queries should be chosen to balance the difficulty between the sub-problems, so that both can be solved.

Tensor queries. In this paper we do not use linear queries because we do not know how to implement linear queries via point queries in the linear-time regime.[6] Nevertheless, we identify a rich-enough sub-class of linear queries for which we are able to solve both of the aforementioned sub-problems: *tensor queries*. These types of linear combinations were used in the sumcheck protocol for tensor codes [Mei13] and also to construct IOPs with proof length approaching witness length [RR20] (the latter work defines an intermediate model that, informally, is an IOP where the verifier is allowed a single tensor query to the witness).

Informally, in a $(\mathbb{F}, k, t)$-tensor IOP, the verifier may specify a round i and a list $(q_0, q_1, \ldots, q_t)$ and then receives as answer the linear combination $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_i \rangle$, where $q_1, \ldots, q_t \in \mathbb{F}^k$ and the 0-th component q_0 of the tensor vector may be a vector of any length defined over $\mathbb{F}$. The other t components must have fixed lengths k. The fixed lengths impose a recursive structure that we will exploit, while the free length accommodates proof messages of varying sizes. For simplicity, in the rest of the technical overview, we will ignore the 0-th component, assuming that all proof messages have the same length (k^t elements in $\mathbb{F}$).

We formalize the notion of a tensor IOP in Section 3. In fact, we formulate a more general notion of IOP where queries belong to a given query class $\mathcal{Q}$, which specifies which (possibly non-linear) functions of the proof messages are "allowed". Via suitable choices of $\mathcal{Q}$, one can recover the notions of point-query IOPs, linear IOPs, tensor IOPs, and more. Our definitions also account for features such as holography and proximity (both used in this paper). We consider the formulation of IOPs with special queries to be a definitional contribution of independent interest that will help the systematic exploration of other query classes.

2.2 From tensor queries to point queries

We discuss the main ideas behind Theorem 3, which provides a transformation that takes as input an IOP with tensor queries and a linear code and outputs an IOP with point queries that has related complexity parameters. (Details of the transformation can be found in Sections 8 to 10.) The main challenge in designing this transformation is that we need to construct an IOP that efficiently simulates a strong class of queries (tensor queries) by using only a weak class of queries (point queries) and the linearity of the given code. Our transformation combines ideas from several works [Mei13; BCGGHJ17; BBHR18; RR20], as we later explain in Remark 2.1.

Now we discuss our transformation. Below we denote by $(\mathbf{P}, \mathbf{V})$ the $(\mathbb{F}, k, t)$-tensor IOP that is given as input to the transformation. The other input to the transformation is a linear code $\mathcal{C}$ over the field $\mathbb{F}$ with rate $\rho = k/n$ and relative distance $\delta = d/n$ (the code's message length k and alphabet $\mathbb{F}$ match parameters in the tensor IOP). We denote by $(\hat{\mathbf{P}}, \hat{\mathbf{V}})$ the point-query IOP that we construct as output. This latter has three parts: (1) a *simulation phase*; (2) a *consistency test*; and (3) a *proximity test*. We summarize each in turn below.

Part 1: simulation phase. The new prover $\hat{\mathbf{P}}$ and new verifier $\hat{\mathbf{V}}$ simulate $\mathbf{P}$ and $\mathbf{V}$, mediating their interaction with two modifications. First, whenever $\mathbf{P}$ outputs a proof string $\Pi \in \mathbb{F}^{k^t}$ that should be sent to $\mathbf{V}$, $\hat{\mathbf{P}}$ sends to $\hat{\mathbf{V}}$ an encoded proof string $\hat{\Pi} := \mathrm{Enc}(\Pi) \in \mathbb{F}^{n^t}$, for an encoding function $\mathrm{Enc}\colon \mathbb{F}^{k^t} \to \mathbb{F}^{n^t}$ that we discuss shortly. Second, whenever $\mathbf{V}$ outputs a tensor query $q_1 \otimes \cdots \otimes q_t$ for one of the proof strings

[6] Bootle et al. [BCGGHJ17] show how to implement the Ideal Linear Commitment (ILC) model in linear time, which is reminiscent of, but distinct from, the linear IOP model. As noted in [BBCGI19], these are reducible to one another, but with losses in parameters. (Applying the transformation of [BCGGHJ17] to an ILC protocol obtained from a linear IOP does *not* preserve linear time.)

Π_i, $\hat{\mathbf{V}}$ forwards this query (as a message) to $\hat{\mathbf{P}}$, who replies with a "short" proof message that contains the answer $\langle q_1 \otimes \cdots \otimes q_t, \Pi_i \rangle \in \mathbb{F}$; then $\hat{\mathbf{V}}$ simply reads this value and returns it to $\mathbf{V}$ as the query answer (so $\hat{\mathbf{V}}$ can continue simulating the execution of $\mathbf{V}$).

Observe that if $\hat{\mathbf{P}}$ really answers each tensor query truthfully in the simulation then $\hat{\mathbf{V}}$ inherits the soundness of $\mathbf{V}$, because in this case the tensor IOP $(\mathbf{P}, \mathbf{V})$ is perfectly simulated. However, a malicious $\hat{\mathbf{P}}$ need not answer each tensor query truthfully. The goal of the consistency test and the proximity test (both described below) is to prevent the prover $\hat{\mathbf{P}}$ from misbehaving. Namely, these additional parts of the point-query IOP $(\hat{\mathbf{P}}, \hat{\mathbf{V}})$ will enable $\hat{\mathbf{V}}$ to check that the values received from $\hat{\mathbf{P}}$ as answers to $\mathbf{V}$'s tensor queries are consistent with the received (encoded) proof strings.

On the encoding function. The encoding function Enc used in the simulation phase must be chosen to facilitate the design of the consistency proximity tests. We choose $\mathrm{Enc}\colon \mathbb{F}^{k^t} \to \mathbb{F}^{n^t}$ to be the encoding function of the t-wise tensor product $\mathcal{C}^{\otimes t}$ of the "base" linear code $\mathcal{C}$, where t matches the parameter in the tensor IOP. The function Enc is derived from the encoding function $\mathrm{enc}\colon \mathbb{F}^k \to \mathbb{F}^n$ of $\mathcal{C}$. Completeness and soundness of $(\hat{\mathbf{P}}, \hat{\mathbf{V}})$ will ultimately work for *any* linear code $\mathcal{C}$. Crucially to our results on linear-time protocols, prior work [DI14] provides linear-time encodable codes with constant rate over any field $\mathbb{F}$, ensuring that Enc is computable in linear time when t is a constant. Such codes achieving the best known parameters are non-systematic.

Checking the simulation phase. In the simulation phase, $\hat{\mathbf{P}}$ has sent several words $\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell \in \mathbb{F}^{n^t}$ that allegedly are codewords in the tensor code $\mathcal{C}^{\otimes t} \subseteq \mathbb{F}^{n^t}$, in which case they encode some proof strings $\Pi_1, \ldots, \Pi_\ell \in \mathbb{F}^{k^t}$. Moreover, $\hat{\mathbf{P}}$ has also claimed that a list of values $(v_q)_{q \in Q}$ are the answers to a corresponding list of tensor queries Q; namely, if $q = (i, q_1, \ldots, q_t)$ then $v_q = \langle q_1 \otimes \cdots \otimes q_t, \Pi_i \rangle$.

Informally, we seek a sub-protocol for $\hat{\mathbf{V}}$ with the following guarantee: (1) if there is a word $\hat{\Pi}_i$ that is far from $\mathcal{C}^{\otimes t}$ then $\hat{\mathbf{V}}$ rejects with high probability; (2) if all words $\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell$ are close to $\mathcal{C}^{\otimes t}$ but one of the answers is inconsistent with the underlying (unique) encoded messages then $\hat{\mathbf{V}}$ also rejects with high probability. A technical contribution of this paper is the design and analysis of such a sub-protocol.

Our sub-protocol is a black-box combination of a consistency test and a proximity test. In the consistency test, the prover $\hat{\mathbf{P}}$ sends, in one round, proof strings that are partial computations of all the tensor queries, and the verifier $\hat{\mathbf{V}}$ leverages these to check that the answers to tensor queries are correct. The consistency test *assumes* that all proof strings are close to certain tensor codes and so, in the proximity test, the prover $\hat{\mathbf{P}}$ and the verifier $\hat{\mathbf{V}}$ interact, over t rounds, in a protocol whose goal is to ensure that all received proof strings are close to the appropriate codes. We now provide more details for each of the two tests.

Part 2: consistency test. For simplicity of exposition, we describe the consistency test for the simple case where there is a *single* tensor query or, more generally, a single "extended" tensor query $q_0 \otimes q_1 \otimes \cdots \otimes q_t \in \mathbb{F}^{\ell \cdot k^t}$ to the "stacking" of all proof strings. Namely, $\hat{\mathbf{P}}$ claims that the stacked word $\hat{\Pi} := \mathrm{Stack}(\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell) \in \mathbb{F}^{\ell \cdot n^t}$ can be decoded to some stacked proof string $\Pi := \mathrm{Stack}(\Pi_1, \ldots, \Pi_\ell) \in \mathbb{F}^{\ell \cdot k^t}$ such that $v = \langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi \rangle$.[7] Below we view Π as a function $\Pi\colon [\ell] \times [k]^t \to \mathbb{F}$, and $\hat{\Pi}$ as a function $\hat{\Pi}\colon [\ell] \times [n]^t \to \mathbb{F}$.

In the special case where the code $\mathcal{C}$ is systematic, the sumcheck protocol for tensor codes [Mei13; RR20] would yield a consistency test that "evaluates" one component of the tensor query at a time. For the general case, where $\mathcal{C}$ can be *any* linear code, we provide a consistency test that consists of a sumcheck-like protocol applied to the "interleaving" of tensor codes. While it is convenient to present the main ideas behind the prover algorithm by speaking of *decoding* (whose cost may exceed our linear-time goal), we stress that the

[7] Extended tensor queries capture tensor queries to specific proof strings: for any desired $i \in [\ell]$, one can choose $q_0 \in \mathbb{F}^\ell$ to be all zeros except for a 1 in the i-th entry so that $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi \rangle = \langle q_1 \otimes \cdots \otimes q_t, \Pi_i \rangle$.

prover need only perform efficient *encoding* operations. We will denote by $(\mathcal{C}^{\otimes t})^\ell$ the ℓ-wise interleaving of the code $\mathcal{C}^{\otimes t}$, where a single symbol of $(\mathcal{C}^{\otimes t})^\ell$ is the concatenation of ℓ symbols of $\mathcal{C}^{\otimes t}$-codewords.

Proof messages. For each $r \in [t]$, the prover $\hat{\mathbf{P}}$ computes and sends words $\{c_r \colon [k] \times [n]^{t-r} \to \mathbb{F}\}_{r \in [t]}$ where c_r is allegedly an interleaved codeword in $(\mathcal{C}^{\otimes t-r})^k$. Intuitively, $c_1, \ldots, c_t$ will help $\hat{\mathbf{V}}$ perform the consistency check that the value $v \in \mathbb{F}$ is the answer to the tensor query $q_0 \otimes q_1 \otimes \cdots \otimes q_t \in \mathbb{F}^{\ell \cdot k^t}$.

- For $r = 1$, the word $c_1 \in (\mathcal{C}^{\otimes t-1})^k$ is derived from $\hat{\Pi} \in \mathcal{C}^{\otimes t}$ via a "fold-then-decode" procedure, which uses the component $q_0 \in \mathbb{F}^\ell$ of the tensor query. For $\gamma \in \mathbb{F}^\ell$, we denote by $\mathrm{Fold}(\hat{\Pi}; \gamma) \colon [n]^t \to \mathbb{F}$ the function $\sum_{i=1}^{\ell} \gamma_i \cdot \hat{\Pi}_i$ (sum the values of $\hat{\Pi} \colon [\ell] \times [n]^t \to \mathbb{F}$ over the domain $[\ell]$ with coefficients determined by γ). Then, $c_1 \in (\mathcal{C}^{\otimes t-1})^k$ is obtained by partially decoding $\mathrm{Fold}(\hat{\Pi}; q_0)$ (by viewing the values of $\mathrm{Fold}(\hat{\Pi}; q_0) \colon [n]^t \to \mathbb{F}$ over the first component $[n]$ of its domain as $\mathcal{C}$-codewords, and decoding them).
- For each subsequent $r \in \{2, \ldots, t\}$, the word c_r is derived via a similar procedure from c_{r-1} and the component $q_{r-1} \in \mathbb{F}^k$ of the tensor query. Namely, c_r is the codeword in $(\mathcal{C}^{\otimes t-r})^k$ obtained by partially decoding $\mathrm{Fold}(c_{r-1}; q_{r-1}) \in \mathcal{C}^{\otimes t-(r-1)}$ over the first component of its domain as above.

Each round reduces the rank by 1 and, in the last round, the word c_t is a fully decoded message vector in $\mathbb{F}^k$. The tensor query answer $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi \rangle$ is the successive folding of Π by components $q_0, \ldots, q_t$. The r-th message c_r is an encoding in $\mathcal{C}^{\otimes t-r}$ of Π after it has been folded up to the r-th component by $q_0, \ldots, q_r$.

Query phase. The verifier $\hat{\mathbf{V}}$ tests the expected relation between messages across rounds at a random point, and that the execution is consistent with the claimed answer value v. Namely, since each round's messages are expected to be partial decodings of foldings of the prior round's messages, for an honest prover $\hat{\mathbf{P}}$ the following equations relate words across rounds:

- for $r = 1$, $\mathrm{enc}(c_1) = \mathrm{Fold}(\hat{\Pi}; q_0)$;
- for each $r \in \{2, \ldots, t\}$, $\mathrm{enc}(c_r) = \mathrm{Fold}(c_{r-1}; q_{r-1})$.

Above, enc is the encoding function for the base linear code $\mathcal{C}$ applied to the first coordinate of a function with domain $[k] \times [n]^{t-r}$ (for some r), and the identity on all other coordinates.

The above equations motivate a natural consistency test for the verifier. Namely, $\hat{\mathbf{V}}$ samples a random tuple $(j_1, \ldots, j_t) \in [n]^t$ and checks all of the relevant equations at this tuple:

- for $r = 1$, $\mathrm{enc}(c_1)(j_1, \ldots, j_t) = \mathrm{Fold}(\hat{\Pi}; q_0)(j_1, \ldots, j_t)$;
- for each $r \in \{2, \ldots, t\}$, $\mathrm{enc}(c_r)(j_r, \ldots, j_t) = \mathrm{Fold}(c_{r-1}; q_{r-1})(j_r, \ldots, j_t)$.

To compute, e.g., $\mathrm{Fold}(c_{r-1}, ; q_{r-1})(j_r, \ldots, j_t)$ and $\mathrm{enc}(c_r)(j_r, \ldots, j_t)$, $\hat{\mathbf{V}}$ makes k queries to c_{r-1} and c_r.

Finally, $\hat{\mathbf{V}}$ checks consistency with the answer value v via the equation $v = \mathrm{Fold}(c_t; q_t)$.

These consistency checks guarantee that when $\hat{\Pi}, c_1, \ldots, c_{t-1}$ are all codewords in their respective codes, then they encode consistent partial computations of the tensor query $q_0 \otimes q_1 \otimes \cdots \otimes q_t \in \mathbb{F}^{\ell \cdot k^t}$ on the message $\Pi \in \mathbb{F}^{\ell \cdot k^t}$ encoded in $\hat{\Pi} \in \mathbb{F}^{\ell \cdot k^t}$. However, we must ensure that $\hat{\mathbf{V}}$ will reject in the case that any of $\hat{\Pi}$, $c_1, \ldots, c_{t-1}$ are far from being codewords. This will be guaranteed by our proximity test.

Part 3: proximity test. We discuss the proximity test, again for the simple case of a *single* tensor query. In the simulation phase the prover $\hat{\mathbf{P}}$ has sent words $\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell$ allegedly in $\mathcal{C}^{\otimes t}$; this means that $\hat{\Pi} := \mathrm{Stack}(\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell)$ is allegedly in $(\mathcal{C}^{\otimes t})^\ell$. In the consistency test the prover $\hat{\mathbf{P}}$ has sent words $c_1, \ldots, c_t$ where c_r allegedly is in $(\mathcal{C}^{\otimes t-r})^k$. The proximity test will ensure that all these words are close to the respective codes.

A reasonable starting point to design such a test is to remember that tensor codes are locally testable [BS06; Vid15; CMS17]: if a word c is Δ-far from $\mathcal{C}^{\otimes t}$ then a random axis-parallel line of c fails to be a

codeword in $\mathcal{C}$ with probability proportional to Δ. Since we wish to test *interleaved* tensor codewords, a natural strategy is to apply the axis-parallel test to a random linear combination of the tested words. This strategy does produce a proximity test, but has two drawbacks. First, a calculation shows that the query complexity is non-trivial only for $t > 2$, while we will design a test that is non-trivial for $t > 1$.[8] Second, the axis-parallel test has poor tradeoffs between query complexity and soundness error.[9] Hence we take a different approach inspired by the proximity test for the Reed–Solomon code in [BBHR18]; at a modest increase in proof length, our test will work for any $t > 1$ (and thereby subsume the prior work in [BCGGHJ17]) and will have better query-soundness tradeoffs.

We now describe our proximity test, which has t rounds of interaction, followed by a query phase.

Interactive phase. In round $r \in [t]$, $\hat{\mathbf{V}}$ sends to $\hat{\mathbf{P}}$ random challenges $\alpha_r, \beta_r \in \mathbb{F}^k$, and $\hat{\mathbf{P}}$ replies with a word $d_r \colon [k] \times [n]^{t-r} \to \mathbb{F}$ (computed as described below) that is allegedly a codeword in $(\mathcal{C}^{\otimes t-r})^k$. Intuitively, for $r \in [t-1]$, the word d_r will be close to a codeword in $\mathcal{C}^{\otimes t-r}$ if and only if c_{r-1} and d_{r-1} are *both* close to codewords in $\mathcal{C}^{\otimes t-(r-1)}$, up to a small error probability.

- In the first round, the word d_1 is derived from $\hat{\Pi}$ via the same "fold-then-decode" procedure that we have already seen. This time, the folding procedure uses the random challenge $\alpha_1 \in \mathbb{F}^\ell$. Then, d_1 is the codeword in $(\mathcal{C}^{\otimes t-1})^k$ obtained by partially decoding $\mathrm{Fold}(\hat{\Pi}; \alpha_1) \in \mathcal{C}^{\otimes t}$.
- In each subsequent round $r = 2, \ldots, t$, the word d_r is derived via a similar procedure from c_{r-1} and d_{r-1}, and the random challenges $\alpha_r, \beta_r \in \mathbb{F}^k$. Namely, d_r is the codeword in $(\mathcal{C}^{\otimes t-r})^k$ obtained by partially decoding $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r) \in \mathcal{C}^{\otimes t-(r-1)}$.

Each round reduces the rank of the tensor code and, in the last round (when $r = t$), the words c_t and d_t are fully decoded message vectors in $\mathbb{F}^k$.

Query phase. The verifier $\hat{\mathbf{V}}$ tests the expected relation between messages across rounds at a random point. Since each round's messages are expected to be partial decodings of foldings of the prior round's messages, for an honest prover $\hat{\mathbf{P}}$ the following equations relate words across rounds:

- for $r = 1$, $\mathrm{enc}(d_1) = \mathrm{Fold}(\hat{\Pi}; \alpha_1)$;
- for each $r \in \{2, \ldots, t\}$, $\mathrm{enc}(d_r) = \mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)$.

As in the consistency test, the above equations motivate natural checks for the verifier. Namely, $\hat{\mathbf{V}}$ samples a random tuple $(j_1, \ldots, j_t) \in [n]^t$ and checks all of the relevant equations at this tuple:

- for $r = 1$, $\mathrm{enc}(d_1)(j_1, \ldots, j_t) = \mathrm{Fold}(\hat{\Pi}; \alpha_1)(j_1, \ldots, j_t)$;
- for each $r \in \{2, \ldots, t\}$, $\mathrm{enc}(d_r)(j_r, \ldots, j_t) = \mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)(j_r, \ldots, j_t)$.

Similarly to before, to obtain the values needed to perform these checks, $\hat{\mathbf{V}}$ makes ℓ point queries to $\hat{\Pi}$ and k point queries to c_r and d_r for each $r \in [t-1]$.

Efficiency. The tensor IOP $(\mathbf{P}, \mathbf{V})$ given as input to the transformation has proof length l, query complexity q, prover arithmetic complexity tp, and verifier arithmetic complexity tv. Now we discuss the main information-theoretic efficiency measures of the constructed point-query IOP $(\hat{\mathbf{P}}, \hat{\mathbf{V}})$.

- *Proof length* is $O_{\rho,t}(\mathsf{q} \cdot \mathsf{l})$. Indeed, in the simulation phase $\hat{\mathbf{P}}$ encodes and sends all the proof strings produced by $\mathbf{P}$, increasing the number of field elements from $\mathsf{l} = \ell \cdot k^t$ to $\ell \cdot n^t = \frac{n^t}{k^t} \cdot \ell \cdot k^t = \rho^{-t} \cdot \mathsf{l}$.

[8] Query complexity for the strategy using local testing would be $O((\ell + kt) \cdot n)$, while that for our test will be $O(\ell + kt)$.

[9] Let $\delta = d/n$ be the relative distance of $\mathcal{C}$. By incurring a multiplicative increase of λ in query complexity, the strategy using local testing gives a soundness error of, e.g., $O(d^t/|\mathbb{F}|) + (1 - \delta^{O(t)} \cdot \Delta)^\lambda$ when applied to an input of distance Δ from $\mathcal{C}^{\otimes t}$. In contrast, the test in this work will give a soundness error that is (approximately) $O(d^t/|\mathbb{F}|) + (1 - \Delta)^\lambda$.

(Plus the q answers to the q tensor queries.) Moreover, in the consistency and proximity tests, $\hat{\mathbf{P}}$ sends, for each of the q queries, $O(n^t)$ field elements in total across t rounds. The sum of these is bounded by $O(\rho^{-t} \cdot \mathsf{q} \cdot \mathsf{l})$.

- *Query complexity* is $O(\ell + t \cdot k \cdot \mathsf{q})$. In the simulation phase, $\hat{\mathbf{V}}$ reads the q answers to the q tensor queries of $\mathbf{V}$ as claimed by $\hat{\mathbf{P}}$. In the consistency and proximity tests, $\hat{\mathbf{V}}$ makes a consistency check that requires point queries on each of the ℓ words $\hat{\Pi}_1, \ldots, \hat{\Pi}_\ell$, plus $O(t \cdot k)$ point queries for each of the q tensor queries. The sum of these is bounded by $O(\ell + t \cdot k \cdot \mathsf{q})$.

Note that the tensor IOP that we construct has query complexity $\mathsf{q} = O(1)$ (see Section 2.4), which means that the multiplicative overheads that arise from the number of queries are constant.

Next we discuss computational efficiency measures. These will depend, in particular, on the cost of encoding a message using the base code $\mathcal{C}$. So let $\theta(k)$ be such that $\theta(k) \cdot k$ is the size of an arithmetic circuit that maps a message in $\mathbb{F}^k$ to its codeword in $\mathcal{C}$. In this paper we focus on the case where $\theta(k)$ is a constant.

- *Verifier arithmetic complexity* is $\mathsf{tv} + O_t((\ell + \theta(k) \cdot k) \cdot \mathsf{q})$. The first term is due to $\hat{\mathbf{V}}$ simulating $\mathbf{V}$ in the simulation phase. In addition, in executing the proximity and consistency tests, $\hat{\mathbf{V}}$ makes, for each of q queries and for each of t rounds, an encoding operation that costs $\theta(k) \cdot k$ plus other linear combinations that cost $O(k)$ field operations, and $O(\ell)$ field operations in the first round. Thus in total, $\hat{\mathbf{V}}$ performs $O((\ell + t \cdot \theta(k) \cdot k) \cdot \mathsf{q})$ field operations in the proximity and consistency tests.

- *Prover arithmetic complexity* is $\mathsf{tp} + O_{\rho,t}(\mathsf{q} \cdot \mathsf{l}) \cdot \theta(k)$. The first term is due to $\hat{\mathbf{P}}$ simulating $\mathbf{P}$ in the simulation phase. In the simulation phase, $\hat{\mathbf{P}}$ also has to encode every proof string output by $\mathbf{P}$. This costs $O(\rho^{-t} \cdot \theta(k) \cdot \mathsf{l})$ field operations, as can be seen by observing that the cost of encoding a single proof string $\Pi_i \in \mathbb{F}^{k^t}$ to its corresponding codeword $\hat{\Pi}_i \in \mathbb{F}^{n^t}$ in $\mathcal{C}^{\otimes t}$ is $O(\rho^{-t} \cdot \theta(k) \cdot k^t)$. Establishing a good bound on the cost of $\hat{\mathbf{P}}$ in the consistency and proximity tests requires more care, as we now explain.

  In the consistency and proximity tests, $\hat{\mathbf{P}}$ must compute each of the functions $c_1, \ldots, c_t$ and $d_1, \ldots, d_t$. Each c_r and d_r is defined in terms of the previous c_{r-1} and d_{r-1} via a "fold-then-decode" procedure. However, we do *not* wish for $\hat{\mathbf{P}}$ to depend on the cost of decoding the base code $\mathcal{C}$, because for the codes that we eventually use ([DI14]), where θ is constant, *no linear-time error-free decoding algorithm is known*. (Only the error-free case need be considered when designing an honest prover algorithm. Indeed, we never use a decoding algorithm of any sort for $\mathcal{C}$ at any point in this work.) Thankfully, since $\hat{\mathbf{P}}$ knows the message Π encoded in $\hat{\Pi}$, $\hat{\mathbf{P}}$ can compute c_r and d_r for each $r \in [t]$ from scratch from Π by *partially re-encoding*, which contributes an additional term of $O(\rho^{-t} \cdot \theta(k) \cdot k^t)$ per query.

Remark 2.1. Our construction of a point-query IOP from a tensor IOP and a linear code builds on several prior works. Below, we highlight similarities and differences with each of these works in chronological order.

- The ILC-to-IOP transformation in [BCGGHJ17] shows how any protocol in the Ideal Linear Commitment (ILC) model can be implemented via a point-query IOP, using any given linear code $\mathcal{C}$ as an ingredient. Crucially, if $\mathcal{C}$ has a linear-time encoding procedure, then computational overheads in the transformation are constant. This is what enables [BCGGHJ17] to obtain a linear-time IOP with square-root query complexity.

  Our construction also relies on an arbitrary linear code $\mathcal{C}$ as an ingredient but considers a different implementation problem (tensor queries via point queries), which ultimately enables much smaller query complexity in the resulting point-query IOP. The interactive phase of our construction could be viewed as a recursive variant of the transformation in [BCGGHJ17].

- The "FRI protocol" in [BBHR18] is an IOP for testing proximity of a function to the Reed–Solomon code. The interactive phase consists of a logarithmic number of rounds in which the proximity problem is reduced in size; the reduction relies on a folding operation defined over subgroups that has small locality, and a low probability of distortion. The query phase consists of a correlated consistency check across all rounds.

  Our proximity test could be viewed as an analogue of the FRI protocol for (the interleaving of) tensor codes. Our consistency test could then be viewed as an analogue of using "rational constraints" and the FRI protocol to check the claimed evaluations of the polynomial committed in a Reed–Solomon codeword.

- The sumcheck protocol for the tensor product of *systematic* codes [Mei13] can simulate a tensor query to a proof string via point queries, via the code-switching technique in [RR20]. This preserves the linear time of the prover, and so could be used to prove Theorem 3 for the special case of a systematic code. Our protocol can be viewed as a non-interactive variant that also works for the *interleaving* of codewords from the tensor product of *non*-systematic codes (as required by Theorem 3). As discussed in Section 1.1, the ability to freely choose any linear code allows better rate-distance tradeoffs and enables the zero-knowledge property to be achieved more efficiently. Further, at the cost of a moderate increase in proof length, our query complexity and verifier complexity scale better with soundness error when doing soundness amplification.[10]

2.3 On soundness of the transformation

The theorem below informally states the security guarantees of the transformation given in the previous section. Details can be found in Section 9 and Section 10. In the rest of this section, we provide some intuition behind the structure of the soundness analysis and the origin of each term in the soundness error.

Theorem 4 (informal)**.** *If $(\mathbf{P}, \mathbf{V})$ is an $(\mathbb{F}, k, t)$-tensor IOP with soundness error ϵ and $\mathcal{C}$ is a linear code with rate $\rho = k/n$ and relative distance $\delta = d/n$, then the point-query IOP $(\hat{\mathbf{P}}, \hat{\mathbf{V}})$ has soundness error*

$$\epsilon + O\left(\frac{d^t}{|\mathbb{F}|}\right) + O\left(\left(1 - \frac{\delta^t}{2}\right)^{\lambda}\right)$$

when the query phases of the consistency and proximity tests are repeated λ times.

The first term ϵ is inherited from soundness of the original protocol; $\hat{\mathbf{P}}$ may attempt to cheat by accurately simulating a cheating $\mathbf{P}$ in a tensor IOP protocol. The remaining terms are upper bounds on the probability that $\hat{\mathbf{V}}$ will accept when the messages of $\hat{\mathbf{P}}$ fail to accurately simulate tensor queries to $\mathbf{P}$'s messages.

The second term is related to a phenomenon of linear codes known as *distortion*. It is important to consider distortion in the soundness analysis of the proximity test. Given interleaved words $W = (w_1, \ldots, w_\ell) \in \mathbb{F}^{\ell \times n}$ with blockwise distance $e := d(W, \mathcal{C}^\ell)$, we use a result from [AHIV17] that shows that the probability that a random linear combination w of $w_1, \ldots, w_\ell$ satisfies $d(w, \mathcal{C}) < e$ (distortion happens) is $O(d/|\mathbb{F}|)$. In other words, for a random linear combination α, $\mathrm{Fold}(\,\cdot\,; \alpha)$ preserves distance with high probability. The term $O(d^t/|\mathbb{F}|)$ in the soundness error comes from bounding the probability of distortion for each code $\mathcal{C}^{\otimes t-r}$ for $r \in [0, \ldots, t-1]$, which has minimum distance d^{t-r}, as $\hat{\mathbf{P}}$ sends and folds words that are allegedly from each of these codes in the proximity test. Combining the distortion result with a union bound gives probability $O((d^t + d^{t-1} + \cdots + d)/|\mathbb{F}|)$ of distortion occurring anywhere in the protocol. The geometric series is asymptotically dominated by its largest term, hence the bound.

[10] Consider the setting in [RR20], which is a single tensor query ($\mathsf{q} = 1$) to a single tensor codeword ($\ell = 1$). The sumcheck protocol in [RR20] branches at each recursion, and has query complexity λ^t and verifier time $\mathsf{poly}(\lambda^t, t, k)$ to achieve soundness error $2^{-\Omega(\lambda)}$. By contrast, we achieve query complexity $O(\lambda \cdot kt)$ and verifier time $O(\lambda \cdot \theta kt)$, where θ is a constant.

The third term comes from the probability that $\hat{\mathbf{V}}$ rejects in the proximity test, given that $\hat{\mathbf{P}}$ sends c_r or d_r which are far from $\mathcal{C}^{\otimes t-r}$ or that $\hat{\mathbf{V}}$ rejects in the consistency test, given that c_r or d_r contain messages which are inconsistent with earlier c and d words. In either case, the fraction of indices on which the verification equations do not hold is then related to the relative distance of $\mathcal{C}^{\otimes t}$, which is δ^t. Here, λ is the number of entries at which $\hat{\mathbf{V}}$ makes the verification checks in the consistency and proximity tests.

The above is an intuitive summary, and in the paragraphs below we elaborate further on our analysis.

Soundness analysis. The proof that our transformation is sound is rather involved, and is a key technical contribution of this paper. The proof is split into two main parts; the analysis of the consistency test and the analysis of the proximity test. The proximity test comprises the most complex part of the analysis.

Our proximity test is recursive, which initially suggests an analysis that recursively applies ideas from [BCGGHJ17]. However, a notable feature of our proximity test is that the verification checks for each $r \in [t]$ are *correlated*. Namely, the verifier $\hat{\mathbf{V}}$ does *not* check e.g. $\mathrm{Fold}\left(c_{r-1}, d_{r-1}; \alpha_r, \beta_r\right) = \mathrm{enc}(d_r)$ for a random point independently of the other verification equations for other values of r. Rather, $\hat{\mathbf{V}}$ samples $(j_1, \ldots, j_t) \in [n]^t$ and checks whether $\mathrm{Fold}(\hat{\Pi}; \alpha_1)(j_1, \ldots, j_t) = \mathrm{enc}(d_1)(j_1, \ldots, j_t)$. Then, for the verification check that e.g. $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_{r-1}, \beta_{r-1}) = \mathrm{enc}(d_r)$, $\hat{\mathbf{V}}$ will truncate $(j_1, \ldots, j_t)$ to $(j_r, \ldots, j_t)$ and check that $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_{r-1}, \beta_{r-1})(j_r, \ldots, j_t) = \mathrm{enc}(d_r)(j_r, \ldots, j_t)$.

We take inspiration from the soundness analysis for the Reed–Solomon proximity test in [BBHR18]. The analysis in [BBHR18] handles their entire protocol with all correlated consistency checks in one single analysis, and avoids a multiplicative dependence on the number of rounds, which was important in [BBHR18] whose protocol had non-constant round-complexity. The same approach informs our analysis, which has the same structure as that of [BBHR18], but is adapted to the combinatorial setting of tensor codes rather than the algebraic setting of Reed–Solomon codes, and modified to reflect the fact that we wish to perform a proximity test for alleged tensor codewords $\hat{\Pi}, c_1, \ldots, c_{t-1}$ of different ranks in the same protocol (rather that one codeword).

Our analysis is divided into cases, depending on the behavior of a malicious prover $\hat{\mathbf{P}}$.

Proximity test soundness. First, suppose that, for some $r \in [t]$, $\hat{\mathbf{P}}$ has sent words c_{r-1} and d_{r-1} that are far from being interleaved $\mathcal{C}^{\otimes t-(r-1)}$-codewords. Yet, through unlucky random choice of α_r or $\beta_r \in \mathbb{F}^k$, one of the intermediate values $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)$ is close to $\mathcal{C}^{\otimes t-(r-1)}$. Then, there exists a valid partial decoding d_r that satisfies consistency checks at a large fraction of entries, potentially causing $\hat{\mathbf{V}}$ to accept even though $\hat{\mathbf{P}}$ has not simulated any inner prover $\mathbf{P}$. Since $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)$ is a random linear combination of words far from $\mathcal{C}^{\otimes t-(r-1)}$, this implies that distortion has occurred. We apply upper bounds on the probability of distortion.

Second, assume that distortion does not occur in any round. Suppose that the prover $\hat{\mathbf{P}}$ has sent c_{r-1} which is far from being an interleaved $\mathcal{C}^{\otimes t-(r-1)}$-codeword. Consider the latest round r for which $\hat{\mathbf{P}}$ behaves in this way. Then $\mathrm{enc}(d_r)$ is close to $\mathcal{C}^{\otimes t-(r-1)}$, but $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)$ is far from $\mathcal{C}^{\otimes t-r}$. Using this fact, the analysis of this case follows from a simpler sub-case. In this sub-case, suppose that $\hat{\mathbf{P}}$ has behaved honestly from the $(r+1)$-th round of the consistency phase onwards, but $\hat{\mathbf{V}}$ makes checks at entries correlated with positions where $\mathrm{Fold}(c_{r-1}, d_{r-1}; \alpha_r, \beta_r)$ is not a $\mathcal{C}^{\otimes t-(r-1)}$-codeword. We show that $\hat{\mathbf{V}}$ will reject.

Consistency test soundness. Suppose that the prover $\hat{\mathbf{P}}$ has sent c_{r-1} that is close to an interleaved codeword, but encodes a message that is not consistent with Π. Consider the latest round r for which $\hat{\mathbf{P}}$ behaves in this way. Then, $\mathrm{enc}(c_r)$ and $\mathrm{Fold}(c_{r-1}; q_{r-1})$ are close to different codewords of $\mathcal{C}^{\otimes t-(r-1)}$. This means that for a large fraction of entries $(j_r, \ldots, j_t) \in [n]^{t-r}$ which is related to the relative distance of the code, $\mathrm{Fold}(c_{r-1}; q_{r-1})(j_r, \ldots, j_t) \neq \mathrm{enc}(c_r)(j_r, \ldots, j_t)$, causing $\hat{\mathbf{V}}$ to reject.

Finally, suppose that, for each $r \in [t]$, $\hat{\mathbf{P}}$ has sent c_r that is an interleaved $\mathcal{C}^{\otimes t-r}$-codeword except for noise at a small number of positions, and all encode messages consistent with queries on Π. In this case, $\hat{\mathbf{P}}$

has essentially simulated an inner prover $\mathbf{P}$ correctly, in the sense that an "error-correction" of the words sent by $\hat{\mathbf{P}}$ are a correct simulation. The soundness error is then inherited from the original protocol $(\mathbf{P}, \mathbf{V})$.

2.4 Checking constraint systems with tensor queries

Our transformation from tensor queries to point queries (Theorem 3) introduces a *multiplicative* blow-up in prover arithmetic complexity (and proof length) that is proportional to the number q of tensor queries. So, for us to ultimately obtain a point-query IOP with linear arithmetic complexity, it is important that the tensor IOP given to the transformation has constant query complexity and a prover with linear arithmetic complexity.

Towards this end, we now turn to Theorem 2, which requires a suitably-efficient tensor IOP for the problem of *rank-1 constraint satisfiability* (R1CS), a generalization of arithmetic circuits given in Definition 1; recall that N is the number of variables and M is the number of coefficients in each coefficient matrix.

A natural starting point would be to build on interactive proofs for evaluating layered arithmetic circuits [GKR08], whose prover can be realized in linear time [Tha13; XZZPS19]. Indeed, the verifier in these protocols only needs to query the low-degree extension of the circuit input, which can be realized via a tensor query to a proof string containing the input sent by the prover. Moreover, the verifier in these protocols is sublinear given oracle access to the low-degree extension of the circuit description. These oracles can be implemented via a sub-protocol if the circuit is sufficiently uniform [GKR08] but, in general, this would require a holographic subprotocol that supports arbitrary circuits (not a goal in those works).

We take a different starting point that is more convenient to describe our holographic tensor IOP for R1CS (and recall that R1CS is a generalization of arithmetic circuits). First, as a warm-up in this section, we discuss a simple construction that fulfills a relaxation of the theorem: a tensor IOP for R1CS with linear proof length $\mathsf{l} = O(N)$, constant query complexity $\mathsf{q} = O(1)$, a prover with linear arithmetic complexity $\mathsf{tp} = O(M)$, and a verifier with linear arithmetic complexity $\mathsf{tv} = O(M)$. After that, in Section 2.5, we describe how to modify this simple protocol to additionally achieve sublinear verification time (incurring only minor losses in the other efficiency parameters). Along the way, we uncover new, and perhaps surprising, connections between prior work on linear-time IOPs [BCGGHJ17] and linear-time sumcheck protocols [Tha13].

In the paragraphs below we denote by $(\mathbf{P}, \mathbf{V})$ the $(\mathbb{F}, k, t)$-tensor IOP that we design for R1CS. We outline its high-level structure, and then describe in more detail the main sub-protocol that enables linear arithmetic complexity, which is for a problem that we call *twisted scalar product* (TSP).

High-level structure. The R1CS problem asks whether, given coefficient matrices $A, B, C \in \mathbb{F}^{N \times N}$ and an instance vector x over $\mathbb{F}$, there exists a witness vector w over $\mathbb{F}$ such that $z := (x, w) \in \mathbb{F}^N$ satisfies $Az \circ Bz = Cz$. Using a similar approach to other proof protocols for R1CS, it suffices for the prover $\mathbf{P}$ to send the full assignment z and its linear combinations $z_A, z_B, z_C \in \mathbb{F}^N$, and convince the verifier $\mathbf{V}$ that $z_A = Az$, $z_B = Bz$, $z_C = Cz$, and $z_A \circ z_B = z_C$ in linear time and using $O(1)$ tensor queries.

To check the first three conditions, the verifier sends a random challenge vector $r \in \mathbb{F}^{n_{\text{row}}}$ with tensor structure. Multiplying on the left by $r^\intercal$ reduces the first three conditions to $\gamma_A = \langle r_A, z \rangle$, $\gamma_B = \langle r_B, z \rangle$, and $\gamma_C = \langle r_C, z \rangle$; here $\gamma_A := \langle r, z_A \rangle$ and $r_A := r^\intercal A$, and similarly for B and C. The verifier can directly obtain the inner products $\gamma_A, \gamma_B, \gamma_C$ through tensor queries to z_A, z_B, z_C. Moreover, both the prover and verifier can locally compute the three vectors r_A, r_B, r_C by right-multiplication by A, B, C respectively, which entails performing a number of arithmetic operations that is linear in the number M of non-zero entries of the matrices.[11] Note that this is the only place where the verifier has to read the entries of A, B, C. The verifier must now check the scalar products $\gamma_A = \langle r_A, z \rangle, \gamma_B = \langle r_B, z \rangle, \gamma_C = \langle r_C, z \rangle$.

[11] We remark that one can improve this cost from linear in the number M of non-zero entries in A, B, C to linear in the cost of right multiplication by A, B, C. By the transposition principle (see Appendix A or, e.g., [KKB88]), this latter is closely related to the cost E of *left* multiplication by A, B, C, which could be much less than M. For example, if A is the matrix corresponding to a

Thus, to check R1CS satisfiability, it suffices to check three scalar products and one Hadamard product. (We must also check that $z = (x, w)$, but this is not the main technical challenge.) We solve both scalar and Hadamard products with a common sub-routine for twisted scalar products that has a linear-time prover and a constant number of tensor queries, as we discuss below. We refer the reader to Section 5 for the details.

Twisted scalar products. The main technical contribution in our tensor IOP construction is the design of a protocol for verifying *twisted scalar products* (TSP).

Definition 3. *The* **twisted scalar product** *of two vectors $u = (u_1, \ldots, u_N)$ and $v = (v_1, \ldots, v_N)$ in $\mathbb{F}^N$ with respect to a third vector $y = (y_1, \ldots, y_N)$ in $\mathbb{F}^N$ is defined to be $\langle u \circ y, v \rangle = \sum_{i=1}^{N} u_i y_i v_i$. In other words, the i-th term $u_i v_i$ contributing to the scalar product $\langle u, v \rangle$ has been multiplied, or "twisted", by y_i.*

Standard scalar products (which we need for $\gamma_A = \langle r_A, z \rangle$, $\gamma_B = \langle r_B, z \rangle$, and $\gamma_C = \langle r_C, z \rangle$) follow by setting $y := 1^N$. To handle the Hadamard product $z_A \circ z_B = z_C$, we pick a random vector y, and up to a small error over the random choice of y, checking the Hadamard product is equivalent to checking the twisted scalar product $\langle u \circ y, v \rangle = \tau$ with $u = z_A$, $v = z_B$ and $\tau = \langle z_C, y \rangle$. In sum, to check the R1CS relation we will check four appropriate instances of the twisted scalar product.

Our result for twisted scalar products is as follows.

Lemma 1 (informal)**.** *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor IOP for twisted scalar products that supports vectors of length $N = k^t$ and twists of the form $y = y_1 \otimes \cdots \otimes y_t$, and has the following parameters:*

- *soundness error is $O(\frac{\log N}{|\mathbb{F}|})$;*
- *round complexity is $O(\log N)$;*
- *proof length is $O(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(1)$;*
- *the prover and verifier both use $O(N)$ field operations.*

Lemma 1 follows from prior work: the linear-time sumcheck of [Tha13; XZZPS19] can be applied to the multi-linear extension of the two vectors in the scalar product, and the verifier's queries to those extensions can be implemented as a tensor query. (The twist can also be handled by "folding it" into a tensor query.)

Below we give an alternative proof inspired by the linear-time protocols of [BCGGHJ17], themselves based on [Gro09]. This is interesting because this latter predates [Tha13] and formerly appeared to be a totally distinct design strategy for interactive protocols. In contrast we show a sumcheck-based protocol inspired by these works, and show that they are a different application of the same linear-time sumcheck. From a technical point of view, our scalar-product protocol invokes the linear-time sumcheck on polynomials that encode information in their coefficients rather than in their evaluations (as is usually the case). This leads to modest opportunities for optimization (see Remark 5.23) and may have applications when used in combination with polynomial commitments not known to support the Lagrange basis (such as [BFS20]). Below we sketch our construction; details are in Section 5.2. For simplicity, below we explain the case of scalar products without a twist. Readers who are comfortable with Lemma 1 may skip the rest of this section.

Strawman construction. Before our construction, we first present a simple linear IOP (an IOP with linear queries as defined in Section 2.1) for scalar products, and then highlight the challenges that we need to overcome to obtain our protocol.

The verifier $\mathbf{V}$ has linear query access to two vectors $u = (u_0, \ldots, u_{N-1})$ and $v = (v_0, \ldots, v_{N-1})$ in $\mathbb{F}^N$. The prover $\mathbf{P}$ wishes to convince the verifier $\mathbf{V}$ that $\langle u, v \rangle = \tau$ for a given $\tau \in \mathbb{F}$. Define the two

discrete Fourier transform, then $E = O(N \log N)$ is much less than $M = \Theta(N^2)$.

polynomials $U(X) := \sum_{i=0}^{N-1} u_i X^i$ and $V(X) := \sum_{i=0}^{N-1} v_i X^{N-i}$ (the entries of v appear in reverse order in $V(X)$). The product polynomial $W(X) := U(X)V(X)$ has $\langle u, v\rangle$ as the coefficient of X^{N-1}, because for any $i, j \in [N]$, the powers of X associated with u_i and v_j multiply together to give X^{N-1} if and only if $i = j$. With this in mind, $\mathbf{P}$ sends to $\mathbf{V}$ the vector $w := (w_0, \ldots, w_{2N-2})$ of coefficients of $W(X)$.

Next, $\mathbf{V}$ checks the equality $W(X) = U(X) \cdot V(X)$ at a random point: it samples a random $\rho \in \mathbb{F}$; constructs the queries $\nu_1 := (1, \rho, \rho^2, \ldots, \rho^{N-1})$, $\nu_2 := (\rho^{N-1}, \rho^{N-2}, \ldots, 1)$, and $\nu_3 := (1, \rho, \rho^2, \ldots, \rho^{2N-2})$; queries u, v, w respectively at ν_1, ν_2, ν_3 to obtain $\gamma_u = \langle u, \nu_1\rangle = U(\rho)$, $\gamma_v = \langle v, \nu_2\rangle = V(\rho)$, $\gamma_w = \langle w, \nu_3\rangle = W(\rho)$; and checks that $\gamma_u \cdot \gamma_v = \gamma_w$. By the Schwarz–Zippel lemma, this is test is unlikely to pass unless $U(X) \cdot V(X) = W(X)$ as polynomials, and in particular, if the coefficient of X^{N-1} in $W(X)$ is not equal to $\langle u, v\rangle$. Finally, $\mathbf{V}$ constructs the query $\nu_4 := (0, \ldots, 1, 0, \ldots, 0)$, which has a 1 in the N-th position of the vector; then queries w at ν_4 to get $w_{N-1} = \langle w, \nu_4\rangle$, and checks that it is equal to τ.

This approach gives a linear IOP for verifying scalar products, with $O(1)$ queries and proof length $O(N)$. One can easily convert it into a linear IOP for verifying twisted scalar products by using $\nu_1 \circ y$ instead of ν_1. With additional care, these queries can even be expressed as tensor queries. However, *the main problem with this approach is that* $\mathbf{P}$ *requires* $O(N \log N)$ *operations to compute* $W(X)$ *by multiplying* $U(X)$ *and* $V(X)$.

Scalar products via sumcheck. We explain how to obtain a tensor IOP for scalar products where $\mathbf{P}$ uses $O(N)$ operations. First, we explain how to redesign the polynomials $U(X)$ and $V(X)$. Then, we explain how to verify that the scalar product is correct via a sumcheck protocol on the product of these polynomials.

We embed the entries of u and $v \in \mathbb{F}^n$ into *multilinear* polynomials $U(X_1, \ldots, X_l)$ and $V(X_1, \ldots, X_l)$ over $\mathbb{F}$. Namely, in $U(X)$, we replace the monomial X^i, which has coefficient u_i, with a monomial in formal variables $X_1, X_2, \ldots, X_{\log N}$, choosing to include X_j if the j-th binary digit of i is a 1. For example, u_0, u_1, u_2 and u_3 are associated with monomials 1, X_1, X_2, and X_1X_2. Thus, each coefficient u_i is associated with a unique monomial in $X_1, \ldots, X_{\log N}$. As with the strawman solution, the coefficients of $V(X)$ are associated with the same monomials, but in reverse order. For example, v_0 and v_1 are associated with monomials $X_1X_2 \cdots X_{\log N}$ and $X_2 \cdots X_{\log N}$. This time, the product polynomial $W(X_1, \ldots, X_{\log N}) := U(X_1, \ldots, X_{\log N}) \cdot V(X_1, \ldots, X_{\log N})$ has $\langle u, v\rangle$ as the coefficient of $X_1X_2 \cdots X_{\log N}$, since for any $i, j \in [N]$ the monomials associated with u_i and v_j multiply together to give $X_1X_2 \cdots X_{\log N}$ if and only if $i = j$.

Now $\mathbf{V}$ has tensor-query access to u and v, and $\mathbf{P}$ must convince $\mathbf{V}$ that $\langle u, v\rangle = \tau$, which now means checking that $\mathrm{Coeff}_{X_1 \cdots X_l}(W) = \tau$. We turn this latter condition into a sumcheck instance, via a new lemma that relates sums of polynomials over multiplicative subgroups to their coefficients; the lemma extends a result in [BCRSVW19] to the multivariate setting.

Lemma 2 (informal)**.** *Let* H *be a multiplicative subgroup of* $\mathbb{F}$ *and* $p(X_1, \ldots, X_l)$ *a polynomial over* $\mathbb{F}$*. Then for every integer vector* $\vec{j} = (j_1, \ldots, j_l) \in \mathbb{N}^l$*,*

$$\sum_{\vec{\omega}=(\omega_1,\ldots,\omega_l)\in H^l} p(\vec{\omega}) \cdot \vec{\omega}^{\vec{j}} = \left(\sum_{\vec{i}+\vec{j} \equiv \vec{0} \bmod |H|} p_{\vec{i}} \right) \cdot |H|^l \ .$$

Above we denote by $p_{\vec{i}}$ *the coefficient of* $X_1^{i_1} \cdots X_l^{i_l}$ *in* p *and denote by* $\vec{\omega}^{\vec{j}}$ *the product* $\omega_1^{j_1} \cdots \omega_l^{j_l}$*.*

Set $H := \{-1, 1\}$, $p := W$, and $\vec{j} := (1, \ldots, 1)$. Since W has degree at most 2 in each variable, the only coefficient contributing to the sum on the right-hand side is the coefficient of $X_1 \cdots X_l$, which is $\langle u, v\rangle$.

In light of the above, the prover $\mathbf{P}$ and the verifier $\mathbf{V}$ engage in a sumcheck protocol for the following claim:

$$\sum_{\vec{\omega}\in\{-1,1\}^l} U(\vec{\omega})V(\vec{\omega}) \cdot \vec{\omega}^{\vec{1}} = \tau \cdot 2^l \ .$$

During the sumcheck protocol, over l rounds of interaction, $\mathbf{V}$ will send random challenges $\rho_1, \ldots, \rho_l$. After the interaction, $\mathbf{V}$ needs to obtain the value $U(\rho_1, \ldots, \rho_l)V(\rho_1, \ldots, \rho_l)$. We show that, in our setting, $\mathbf{V}$ can obtain the two values in this product by making tensor queries to u and v, respectively.

We are left to discuss how $\mathbf{P}$ can be implemented in $O(2^l) = O(N)$ operations.

Recall that the problem in the strawman protocol was that $\mathbf{P}$ had to multiply two polynomials of degree N. Now the problem seems even worse: $\mathbf{P}$ cannot compute W directly as it has a super-linear number of coefficients (W is multi-quadratic in $l = \log N$ variables). However, in the sumcheck protocol, $\mathbf{P}$ *need not compute and send every coefficient of* W and can compute the messages for the sumcheck protocol by using partial evaluations $U(\rho_1, \ldots, \rho_j, X_{j+1}, \ldots, X_{\log N})$ and $V(\rho_1, \ldots, \rho_j, X_{j+1}, \ldots, X_{\log N})$ without ever performing any high-degree polynomial multiplications. This is indeed the logic behind techniques for implementing sumcheck provers in linear time, as discussed in [Tha13; XZZPS19], which, e.g., suffice for sumchecks where the addend is the product of constantly-many multilinear polynomials, as is the case for us.

The full details, which give explicit tensor queries for evaluating U and V, and how to incorporate the "twist" with $y \in \mathbb{F}^N$ into the sumcheck to get our TSP protocol, are given in Section 5.2.

Remark 2.2 (binary fields)**.** The astute reader may notice that setting $H = \{-1, 1\}$ in Lemma 2 is only possible when the characteristic of $\mathbb{F}$ is not equal to 2. Nevertheless, a statement similar to Lemma 2 holds for additive subgroups, which in particular we can use in the case of binary fields. As we explain in Remark 5.9, our results then carry over with minor modifications to binary fields as well (and thus all large-enough fields).

2.5 Achieving holography

Thus far, we have discussed ingredients behind a relaxation of Theorem 1 with no sublinear verification. Namely, (1) an IOP with tensor queries where the verifier receives as *explicit* input the R1CS coefficient matrices A, B, C; and (2) a transformation from this tensor-query IOP to a corresponding point-query IOP.

We now describe how to additionally achieve the sublinear verification in Theorem 1 via holography.

In a holographic IOP for R1CS, the verifier $\mathbf{V}$ no longer receives as explicit input A, B, C. Instead, in addition to the prover $\mathbf{P}$ and the verifier $\mathbf{V}$, a holographic IOP for R1CS includes an additional algorithm, known as the *indexer* and denoted by $\mathbf{I}$, that receives as explicit input A, B, C and outputs an "encoding" of these. The verifier $\mathbf{V}$ then has query access to the output of the indexer $\mathbf{I}$. This potentially enables the verifier $\mathbf{V}$ to run in time that is sublinear in the time to read A, B, C.

Achieving such a verifier speed-up and thereby obtaining Theorem 1, however, requires modifications in both of the aforementioned ingredients. Below we first discuss the modifications to the transformation, as they are relatively straightforward. After that we dedicate the rest of the section to discuss the modifications to the tensor-query IOP, because making it holographic requires several additional ideas.

Preserving holography in the transformation. Informally, we want the modified transformation to "preserve holography": if the tensor-query IOP given to the transformation is holographic (the verifier has tensor-query access to the output of an indexer), then the new point-query IOP produced by the transformation is also holographic (the new verifier has point-query access to the output of the new indexer). Moreover, the transformation should introduce only constant multiplicative overheads in the cost of indexing and proving.

So let $\mathbf{I}$ be the indexer of the tensor-query IOP. The new indexer $\hat{\mathbf{I}}$ for the point-query IOP simulates $\mathbf{I}$ and encodes its output using Enc, just as the new prover $\hat{\mathbf{P}}$ encodes the messages from $\mathbf{P}$. (Recall from Section 2.2 that Enc is the encoding function for the tensor code $\mathcal{C}^{\otimes t}$.) Subsequently, in the simulation phase of the transformation, whenever the verifier $\mathbf{V}$ wishes to make a tensor query to the output of $\mathbf{I}$, the new verifier $\hat{\mathbf{V}}$ forwards this query to the new prover $\hat{\mathbf{P}}$, who responds with the answer. After that, we extend the consistency and proximity tests in the transformation to also ensure that $\hat{\mathbf{P}}$ answers these additional tensor

queries correctly. These tests will require the new verifier $\hat{\mathbf{V}}$ to make point queries to the encoding of the output of $\mathbf{I}$, which is precisely what $\hat{\mathbf{V}}$ has query access to because that is the output of $\hat{\mathbf{I}}$. The constant multiplicative overheads incurred by the transformation still hold after these (relatively minor) modifications.

A holographic tensor IOP. In the non-holographic tensor-query IOP outlined in Section 2.4, the verifier $\mathbf{V}$, receives as input coefficient matrices A, B, C explicitly, and must perform two types of expensive operations based on these. First, $\mathbf{V}$ expands some random challenges $r_1, \ldots, r_t \in \mathbb{F}^k$ into a vector $r = r_1 \otimes \cdots \otimes r_t \in \mathbb{F}^{k^t}$, which requires $O(k^t)$ arithmetic operations. Second, $\mathbf{V}$ computes the matrix-vector product $r_A := r^\intercal A$, which in the worst case costs proportionally to the number of non-zero entries of A. Similarly for B and C.

Viewed at a high level, these expensive operations are performed as part of a check that $z_A = Az$ (and similarly for B and C), which has been referred to as a "lincheck" (see e.g. [BCRSVW19]). Thus, it is sufficient to provide a "holographic lincheck" subprotocol where $\mathbf{V}$ has tensor query access to a matrix U (which is one of A, B, or C), an input vector v, and an output vector v_{U}, and wishes to check that $v_{\mathrm{U}} = Uv$.

Challenges to holography. To illustrate the challenges of obtaining a linear-time holographic lincheck, we first present a simple strawman: a sound protocol that falls (far) short of linear arithmetic complexity. First we describe the indexer, and after that the interaction between the prover and verifier.

- *Indexer.* The indexer receives as input a matrix U over $\mathbb{F}$, which for simplicity we assume has dimension $k^t \times k^t$; we can then identify the rows and columns of U via tuples $(i_1, \ldots, i_t) \in [k]^t$ and $(j_1, \ldots, j_t) \in [k]^t$ respectively. The indexer outputs the vector $u \in \mathbb{F}^{k^{2t}}$ such that $u_{i_1,\ldots,i_t,j_1,\ldots,j_t}$ is the entry of U at row $(i_1, \ldots, i_t)$ and column $(j_1, \ldots, j_t)$. The verifier will have $(\mathbb{F}, k, 2t)$-tensor-query access to u.

- *Prover and verifier.* To check that $v_{\mathrm{U}} = Uv$, for the verifier it suffices to check that $\langle r, v_{\mathrm{U}} \rangle = \langle r^\intercal U, v \rangle$ for a random $r = r_1 \otimes \cdots \otimes r_t$ in $\mathbb{F}^{k^t}$ (up to a small error over the choice of r). Since $r^\intercal U v = \langle r \otimes v, u \rangle$, the verifier wishes to check whether $\langle r, v_{\mathrm{U}} \rangle = \langle r \otimes v, u \rangle$. The verifier makes the $(\mathbb{F}, k, t)$-tensor query r to v_{U} to obtain the left hand side. To help the verifier obtain the right hand side, the prover computes $e := r \otimes v \in \mathbb{F}^{2t}$ and sends it to the verifier. Since u need not have a tensor structure, the verifier cannot directly obtain $\langle e, u \rangle$ via a $(\mathbb{F}, k, 2t)$-tensor query to e; instead, the verifier can receive this value from the prover and rely on a scalar-product protocol to check its correctness. The verifier is thus left to check that indeed $e = r \otimes v$. Note that for any $s = s_1 \otimes \cdots \otimes s_t$ and $s' = s'_1 \otimes \cdots \otimes s'_t$ it holds that $\langle s \otimes s', e \rangle = \langle s, r \rangle \langle s', v \rangle = \langle s_1, r_1 \rangle \cdots \langle s_t, r_t \rangle \langle s', v \rangle$. The verifier checks this equality for random s and s': it directly computes $\langle s_i, r_i \rangle$ for each $i \in [t]$; obtains $\langle s', v \rangle$ via a $(\mathbb{F}, k, t)$-tensor query to v; obtains $\langle s \otimes s', e \rangle$ via a $(\mathbb{F}, k, 2t)$-tensor query to e; and checks the expression.

Crucially, in the protocol described above, the verifier performs only $O(kt)$ field operations. In particular, the verifier did not have to incur the cost of reading the matrix U, which is much greater in general.

However, the foregoing protocol falls (far) short of achieving linear proving time: the indexer outputs a vector u that specifies the matrix U via a *dense* representation of k^{2t} elements, and this leads to the prover having to compute vectors such as $e \in \mathbb{F}^{2t}$, which costs $O(k^{2t})$ operations. On the other hand, in order to represent U, it suffices to specify its *non-zero entries*. Hence, unless U is a dense matrix (with $\Omega(k^{2t})$ non-zero entries), the foregoing protocol does *not* have a linear-time prover (and also does not have a linear-time indexer).

Our goal here is thus a holographic protocol that is efficient relative to a *sparse* representation of the matrix U, for example the triple of vectors $(\mathrm{val}_U, \mathrm{row}_U, \mathrm{col}_U) \in \mathbb{F}^M \times [k^t]^M \times [k^t]^M$ such that val_U is a list of the values of the M non-zero entries of U, and $\mathrm{row}_U, \mathrm{col}_U$ are the indices of these entries in U (i.e., for all $\kappa \in [M]$ it holds that $U(\mathrm{row}_U(\kappa), \mathrm{col}_U(\kappa)) = \mathrm{val}_U(\kappa)$). This is (essentially) the best that we can hope for, as the indexer and the prover must read a description of U.

Efficiency relative to the sparse representation was achieved in [CHMMVW20; COS20], which contributed efficient holographic IOPs for R1CS. However, the prover algorithm in those constructions runs in quasilinear time, and we do not know how to adapt the techniques in these prior works, which are based on univariate polynomials, to our setting (linear-time tensor IOPs). It remains an interesting open problem to build on those techniques to achieve a linear-time holographic lincheck with tensor queries.

Subsequently, Setty [Set20] constructed a preprocessing SNARG for R1CS without FFTs by porting the univariate protocols for R1CS to the multivariate setting (where we have a linear-time sumcheck [Tha13]) and solving the matrix sparsity problem by constructing a polynomial commitment scheme for "sparse multivariate polynomials". His approach to sparsity is combinatorial rather than algebraic: he constructs a linear-size circuit using memory-checking ideas to "load" each non-zero term of the polynomial and add it to a total sum, and then invokes an argument system for uniform circuits that does not use FFTs [WTSTW18]. Since a key component of the construction is a polynomial commitment scheme for (dense) multilinear extensions, and the multilinear extension of a vector is a special case of a tensor query, it is plausible that one could distill a tensor IOP from [Set20] that suits our purposes. However, taking this path is not straightforward given the protocol's complexity, and the informal discussions and proof sketches in [Set20].

Our approach. To prove our theorem, we build on an earlier protocol of Bootle et al. [BCGJM18] and a recent simplification by Gabizon and Williamson [GW20]. As described below, this leads to a direct and natural construction for a holographic lincheck, which is what we need. The key component in our construction, like the earlier works, is a *look-up protocol*, wherein the prover can convince the verifier that previously-stored values are correctly retrieved. Below we describe how to obtain the lincheck protocol given the look-up protocol as a subroutine, and after that describe the look-up protocol.

As in the strawman protocol, to check that $v_{\mathrm{U}} = Uv$, for the verifier it suffices to check that $\langle r, v_{\mathrm{U}} \rangle = \langle r^{\intercal}U, v \rangle$ for a random $r = r_1 \otimes \cdots \otimes r_t$ in $\mathbb{F}^{k^t}$ (up to a small error over r). Again, the verifier can directly obtain the value $\langle r, v_{\mathrm{U}} \rangle$ by querying v_{U} at the tensor r. The verifier is left to obtain the value of $\langle r^{\intercal}U, v \rangle = r^{\intercal}Uv$, and enlists the prover's help to do so. Therefore, the verifier sends $r_1, \ldots, r_t$ to the prover, who replies with $\gamma := r^{\intercal}Uv \in \mathbb{F}$. The prover must now convince the verifier that γ is correct.

Towards this, the prover will send partial results in the computation of γ, and the verifier will run sub-protocols to check the correctness of each partial result. To see how to do this, we first re-write the expression $r^{\intercal}Uv$ in terms of the sparse representation of U:

$$r^{\intercal}Uv = \sum_{\kappa \in [M]} \mathrm{val}_U(\kappa) \cdot r(\mathrm{row}_U(\kappa)) \cdot v(\mathrm{col}_U(\kappa)) \ . \tag{1}$$

This expression suggests the prover's first message to the verifier, which consists of the following two vectors:

$$r^* := \Big(r(\mathrm{row}_U(\kappa))\Big)_{\kappa \in [M]} \quad \text{and} \quad v^* := \Big(v(\mathrm{col}_U(\kappa))\Big)_{\kappa \in [M]} \ .$$

Observe that, if the prover was honest, then the right-hand side of Equation (1) is equal to $\langle \mathrm{val}_U, r^* \circ v^* \rangle$.

Therefore, the verifier is left to check that: (1) $\gamma = \langle \mathrm{val}_U, r^* \circ v^* \rangle$, and (2) r^*, v^* were correctly assembled from the entries of r, v as determined by the indices in $\mathrm{row}_U, \mathrm{col}_U$, respectively.

The verifier can check the first condition via a scalar product subprotocol and a Hadamard product subprotocol (which we have discussed in Section 2.4). To check the second condition, the verifier will use a *tensor consistency test* and two *look-up subprotocols*, as we now explain.

Even though the verifier sampled the components $r_1, \ldots, r_t \in \mathbb{F}^k$ that determine the tensor vector $r = r_1 \otimes \cdots \otimes r_t \in \mathbb{F}^{k^t}$, the verifier cannot afford to directly compute r, as this would cost $O(k^t)$ operations. Instead, the prover sends r, and the verifier checks that r was computed correctly from $r_1, \ldots, r_t$ via a

simple subprotocol, which we describe in Section 6.2, that only requires making one tensor query to r and performing $O(tk)$ operations. Now the verifier is in a position to make tensor queries to (the correct) r.

Next, observe that r^* is correct if and only if, for each $\kappa \in [M]$, there is $i \in [k^t]$ such that $(r^*_\kappa, \mathrm{row}_U(\kappa)) = (r_i, i)$. We represent this "look-up" condition via the shorthand $(r^*, \mathrm{row}_U) \subseteq (r, [k^t])$. Similarly, v^* is correct if and only if $(v^*, \mathrm{col}_U) \subseteq (v, [k^t])$. We now discuss a protocol to check such conditions.

Look-ups via tensor queries. A look-up protocol is to check the condition $(c, I) \subseteq (d, [k^t])$, given that the verifier has tensor-query access to the vectors $c \in \mathbb{F}^M$ and $d \in \mathbb{F}^{k^t}$, and also to the index vectors $I \in \mathbb{F}^M$ and $[k^t] \in \mathbb{F}^{k^t}$. (Here we are implicitly associating the integers in $[k^t]$ with an arbitrary k^t-size subset in $\mathbb{F}$.)

Look-up protocols for use in proof systems originate in a work of Bootle et al. [BCGJM18] that aims at low computational overhead for the prover. Their protocol reduces the look-up condition to a univariate polynomial identity, and then the prover helps the verifier to check that the polynomial identity holds when evaluated at a random challenge point. However, the prover in their protocol incurs a logarithmic overhead in the size of the list c, which in our usage would result in a superlinear-time prover.

Gabizon and Williams [GW20] provide a more efficient look-up protocol, which removes the logarithmic overhead by relying on a more expressive *bivariate* polynomial identity. However, they use a different proof model that we do not know how to compile into ours while preserving linear complexity. Our contribution is to give a linear-time tensor IOP for look-ups using their polynomial identity as a starting point.

Below we recall the identity and then summarize our look-up protocol; details can be found in Section 6.5.

First recall that, via a standard use of randomized hashing, we can replace the lookup condition $(c, I) \subseteq (d, [k^t])$ with a simpler *inclusion* condition $a \subseteq b$ for suitable vectors $a \in \mathbb{F}^M$ and $b \in \mathbb{F}^{k^t}$ (each entry in the vector a equals some entry of the vector b). The polynomial identity from [GW20] concerns this latter condition, as we now explain. (We also note that we have modified the polynomial identity to incorporate "wrap-around" in the entries of vectors, to simplify other aspects of our protocols.)Assuming for simplicity that the entries of b are *distinct*, let $\mathrm{sort}()$ denote the function that sorts the entries of its input in the order $b_1 \prec b_2 \prec \ldots \prec b_{k^t}$.[12] Let $\mathrm{shift}()$ denote a cyclic shift.

Lemma 2.3 ([GW20])**.** *Let $a \in \mathbb{F}^M$ and $b \in \mathbb{F}^{k^t}$. Then $a \subseteq b$ if and only if there is $w \in \mathbb{F}^{k^t+M}$ such that*

$$\prod_{j=1}^{M+k^t} \big(Y(1+Z) + w_j + \mathrm{shift}(w)_j \cdot Z\big) = (1+Z)^M \prod_{j=1}^{M} (Y + a_j) \prod_{j=1}^{k^t} \big(Y(1+Z) + b_j + \mathrm{shift}(b)_j \cdot Z\big) \ . \quad (2)$$

In the case that $a \subseteq b$, we may take $w = \mathrm{sort}(a, b)$ to satisfy the above equation.

In our look-up protocol, the prover recomputes this polynomial identity at random evaluation points chosen by the verifier and sends intermediate computation steps to the verifier.[13] Both parties run subprotocols to check that the computation was performed correctly and the evaluated polynomial identity holds.

Having sent w to the verifier, the prover also sends the vectors $w_\circlearrowleft := \mathrm{shift}(w)$ and $b_\circlearrowleft := \mathrm{shift}(b)$. Then after receiving random evaluation points for Y and Z, the prover sends vectors w^*, a^*, b^* containing each evaluated term in the first, second, and third products of Equation (2) to the verifier, along with the values $\chi_{w^*} := \mathrm{prod}(w^*), \chi_{a^*} := \mathrm{prod}(a^*), \chi_{b^*} := \mathrm{prod}(b^*)$ of each product as non-oracle messages. Here, $\mathrm{prod}()$ denotes the function which takes the product of the entries of its input.

[12] When the entries of b are not distinct, one can consider a more complex merge operation; see Section 6.5.

[13] One can draw parallels between the combination of randomized hashing and the polynomial identity used in this work, and the combination of randomized and multi-set hashing used in the memory-checking circuit of [Set20]. Conceptually, the [GW20] polynomial identity enforces stronger conditions on w, a and b than a multi-set hash and removes the need for the time-stamping data used in [Set20].

Apart from simple checks that the vectors w^*, a^*, b^* were correctly computed, and using $\chi_{w^*}, \chi_{a^*}, \chi_{b^*}$ to check that Equation (2) holds, we rely on two additional subprotocols.

- A *cyclic-shift* test to show that e.g. $b_{\circlearrowleft} = \mathrm{shift}(b)$. The polynomial identity $\sum_{i=1}^{k^t} b(i)X^{i-1} - X \cdot \sum_{i=1}^{k^t} b_{\circlearrowleft}(i)X^{i-1} = (1 - X^{k^t}) \cdot b_{\circlearrowleft}(k^t)$ holds if and only if $b_{\circlearrowleft} = \mathrm{shift}(b)$. The verifier uses tensor queries to check that this identity holds at a random point. More details can be found in Section 6.3.
- An *entry-product* protocol to show that e.g. $\chi_{w^*} = \mathrm{prod}(w^*)$. This protocol combines a cyclic-shift test with a Hadamard-product protocol in order to verify the correct computation of all partial products leading to the entry product. More details can be found in Section 6.4.

3 Interactive oracle proofs with special queries

Interactive Oracle Proofs (IOPs) [BCS16; RRR16] are information-theoretic proof systems that combine aspects of Interactive Proofs [Bab85; GMR89] and Probabilistically Checkable Proofs [BFLS91; FGLSS91; AS98; ALMSS98], and also generalize the notion of Interactive PCPs [KR08].

The main result in this paper is an IOP protocol but, on the way to this result, we also consider protocols in a variant of the IOP model where "point queries" by the verifier to prover messages are replaced with "tensor queries". In this section we provide a single definition of *IOPs with special queries* that can be used to capture both standard IOPs (with point queries) and the variant that we consider (with tensor queries), as well as other variants in the literature. We consider the definition in this section to be of independent interest.

The definition below specifies the syntax of the proof system, and additionally considers the feature of *holography* (see, e.g., [CHMMVW20; COS20]). Informally, this means that the proof system works for indexed (ternary) relations instead of standard (binary) relations.

Definition 3.1. *An* **indexed relation** R *is a set of triples* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *where* $\mathbb{i}$ *is the index,* $\mathbb{x}$ *the instance, and* $\mathbb{w}$ *the witness. The corresponding* **indexed language** $L(R)$ *is the set of pairs* $(\mathbb{i}, \mathbb{x})$ *for which there exists a witness* $\mathbb{w}$ *such that* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$*.*

For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathbb{i}$ is the description of a boolean circuit, $\mathbb{x}$ is an assignment some of the input wires, and $\mathbb{w}$ is an assignment to the remaining input wires that makes the circuit output 0. In this paper we build protocols for the indexed relation that represents *rank-1 constraint satisfiability* (see Definition 4.1), a generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices.

Definition 3.2. *A* **holographic IOP with query class** $\mathcal{Q}$ *(some set of functions) is a tuple* $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$*, where* $\mathbf{I}$ *is the* indexer*,* $\mathbf{P}$ *the* prover*, and* $\mathbf{V}$ *the* verifier*. The indexer is a deterministic polynomial-time algorithm, while the prover and verifier are probabilistic polynomial-time interactive algorithms.*

In an offline phase, the indexer $\mathbf{I}$ *is given an index* $\mathbb{i}$ *and outputs an encoding* Π_0 *of* $\mathbb{i}$*.*

In an online phase, the prover $\mathbf{P}$ *receives as input an index-instance-witness tuple* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *and the verifier* $\mathbf{V}$ *receives as input the instance* $\mathbb{x}$*; in addition, the verifier* $\mathbf{V}$ *has query access to* Π_0 *(in a precise sense specified below), which we denote as* $\mathbf{V}^{\Pi_0}(\mathbb{x})$*. The online phase consists of multiple rounds, and in each round the prover* $\mathbf{P}$ *sends a proof message* Π_i *and the verifier* $\mathbf{V}$ *replies with a challenge message* ρ_i*.*

The prover $\mathbf{P}$ *may compute its proof message* Π_i *based on its input* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *and all the verifier challenges received thus far (none if* $i = 1$ *or* $\rho_1, \ldots, \rho_{i-1}$ *if* $i > 1$*). In contrast, the verifier* $\mathbf{V}$ *may compute its challenge message* ρ_i *based on its input* $\mathbb{x}$ *and on answers obtained by querying* $(\Pi_0, \Pi_1, \ldots, \Pi_i)$ *via queries in* $\mathcal{Q}$*. In more detail, the answer of a query* $q \in \mathcal{Q}$ *to* $(\Pi_0, \Pi_1, \ldots, \Pi_i)$ *is* $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i)$ *(this answer could also be a special error value in case the proof messages are not according to an expected format).*

After the interaction and all queries are concluded, the verifier $\mathbf{V}$ *accepts or rejects.*

Remark 3.3 (non-oracle messages)**.** We allow the prover in an IOP to also send, at any point in the interaction, arbitrary messages that the verifier will simply read in full (without making any queries) as in a usual interactive proof. We refer to such messages as *non-oracle messages*, to differentiate them from the *oracle messages* to which the verifier has query access. These non-oracle messages can typically be viewed as degenerate cases of oracle messages, and we use them in protocol descriptions for convenience of exposition.

A holographic interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ for an indexed relation R has completeness 1 and soundness error ϵ if the following holds.

- **Completeness.** For every index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept is 1.
- **Soundness.** For every index-instance tuple $(\mathbb{i}, \mathbb{x}) \notin L(R)$ and malicious prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept is at most ϵ.

We consider several complexity measures:

– *round complexity* rc is the number of back-and-forth message exchanges between the prover and verifier;
– *proof length* $\mathsf{l} = \mathsf{li} + \mathsf{lp} + \mathsf{lc}$ where $\mathsf{li} := |\Pi_0|$ is the number of alphabet symbols output by the indexer, $\mathsf{lp} := |\Pi_1| + \cdots + |\Pi_{\mathsf{rc}}|$ is the total number of alphabet symbols sent in oracle messages by the prover, and lc is the total number of alphabet symbols sent in non-oracle messages by the prover;
– *query complexity* q is the total number of queries made by the verifier (to any oracle);
– *running time* ti is the running time of $\mathbf{I}$, tp is the running time of $\mathbf{P}$, and tv is the running time of $\mathbf{V}$.

Public coins. We say that IOP is *public-coin* if each verifier message to the prover is a random string. This means that the verifier's randomness is its challenge messages $\rho_1, \ldots, \rho_{\mathsf{rc}}$. All verifier queries can be postponed, without loss of generality, to a query phase that occurs after the interactive phase with the prover.

Non-adaptive queries. We say that IOP is *non-adaptive* if all verifier queries depend solely on the input instance $\mathbb{x}$ and the verifier's randomness (as opposed to some queries depending on answers to prior queries).

Query classes of interest. We illustrate how to use the notion of IOPs with specials queries to obtain several notion of IOPs, including those that we use in this paper. We begin with IOPs with point queries (i.e., standard IOPs), as expressed in the following definition.

Definition 3.4. *A* **holographic IOP with point queries** *is an IOP with the query class $\mathcal{Q}_{\mathrm{point}}$ defined as follows: $\mathcal{Q}_{\mathrm{point}}$ is all functions of the form $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i) = \Pi_j[k]$ for some $j \in \{0, 1, \ldots, i\}$ and location k. (If the location k does not exist, the answer is an error.) Namely, each query in the class $\mathcal{Q}_{\mathrm{point}}$ returns the symbol at a location of the encoded index ($j = 0$) or of a specified prover message ($j > 0$).*

Next, we define holographic IOPs with linear queries, which generalize the notion of "fully-linear IOPs" used in [BBCGI19] (the generalization is that in [BBCGI19] the indexer is degenerate, i.e., $\mathbf{I}(\mathbb{i}) = \mathbb{i}$).

Definition 3.5. *Given a finite field $\mathbb{F}$, a* **holographic IOP with $\mathbb{F}$-linear queries** *is an IOP with the query class $\mathcal{Q}_{\mathrm{linear}}(\mathbb{F})$ defined as follows: $\mathcal{Q}_{\mathrm{linear}}(\mathbb{F})$ contains all functions of the form $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i) = \langle a, \Pi_j \rangle$ for some $j \in \{0, 1, \ldots, i\}$ and $\mathbb{F}$-linear combination a. (If the lengths of the linear combination a and proof string Π_j do not match, the answer is an error.) Namely, each query in the class $\mathcal{Q}_{\mathrm{linear}}$ returns the scalar product of a vector and the encoded index ($j = 0$) or of a specified prover message ($j > 0$).*

We do not use linear queries in this paper, but we find them a useful comparison to understand the query class that we do use, tensor queries, which we define next.

Definition 3.6. *Given a finite field $\mathbb{F}$ and positive integers k, t, a* **holographic IOP with $(\mathbb{F}, k, t)$-tensor queries** *is an IOP with the query class $\mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ defined as follows: $\mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ contains all functions of the form $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i) = \langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_j \rangle$ for some $j \in \{0, 1, \ldots, i\}$ and vectors q_0 over $\mathbb{F}$ and $q_1, \ldots, q_t \in \mathbb{F}^k$. (If the lengths of the linear combination $q_0 \otimes q_1 \otimes \cdots \otimes q_t$ and proof string Π_j do not match, the answer is an error.) Namely, each query in the class $\mathcal{Q}_{\mathrm{tensor}}$ returns the scalar product of a certain tensor vector and the encoded index ($j = 0$) or of a specified prover message ($j > 0$).*

Observe that tensor queries are a generalization of point queries (every point query can be represented as a tensor vector) and are a restriction of linear queries (every tensor query can be presented as a linear query). In this paper we construct an efficient IOP with tensor queries for constraint systems, and additionally compile any IOP with tensor queries into an IOP with point queries while preserving complexity of the prover.

Remark 3.7. In the context of tensor IOPs, we often view a proof message $\Pi \in \mathbb{F}^{\ell \cdot k^t}$ as consisting of ℓ "sub-messages" $\pi_1, \ldots, \pi_\ell$ each in $\mathbb{F}^{k^t}$. In this case, when describing a protocol, we may make a tensor query $q_1 \otimes \cdots \otimes q_t$ to one of the sub-messages π, with the understanding that one can specify an indicator vector $q_0 \in \mathbb{F}^\ell$ so that $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi \rangle = \langle q_1 \otimes \cdots \otimes q_t, \pi \rangle$.

3.1 Proximity proofs

In this paper we also use, on the way to our main result, proximity proofs, i.e., protocols whose goal is not to show that a valid witness exists but, rather, is to show that a particular string is close to a valid witness. This is captured by the notion of *proximity soundness* below, which strengthens the regular soundness notion. Note that the definition that we use is given with respect to a general distance notion, as opposed to the usual relative hamming distance; this is because the relation that we consider has multi-part witnesses over different-sized domains (Definition 8.4), which requires a distance-metric that takes this into account (Definition 8.5).

In more detail, a *holographic interactive oracle proof of proximity* $\mathsf{IOPP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ for an indexed relation R has completeness 1 and soundness error ϵ with distance function Δ if the following holds.

- **Completeness.** For every index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i}), \mathbb{w}}(\mathbb{x})$ to accept is 1.
- **Proximity soundness.** For every index-instance tuple $(\mathbb{i}, \mathbb{x})$ and malicious prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i}), \mathbb{w}}(\mathbb{x})$ to accept is at most $\epsilon(\Delta(\mathbb{w}, R|_{(\mathbb{i}, \mathbb{x})}))$. Here the soundness error ϵ is a function of the Δ-distance of $\mathbb{w}$ to the set of valid witnesses $R|_{(\mathbb{i}, \mathbb{x})} := \{\mathbb{w}' \mid (\mathbb{i}, \mathbb{x}, \mathbb{w}') \in R\}$. (Intuitively, one expects the soundness error to go down as the distance of $\mathbb{w}$ to valid witnesses increases.)

We remark that, while not explicit above, each verifier's query $q \in \mathcal{Q}$ is to $(\mathbf{I}(\mathbb{i}), \mathbb{w}, \Pi_1, \ldots, \Pi_i)$ and its answer is $q(\mathbb{x}, \mathbf{I}(\mathbb{i}), \mathbb{w}, \Pi_1, \ldots, \Pi_i)$; namely, the candidate witness $\mathbb{w}$ is also an input to the query function q. The definition of IOPs with particular query classes (e.g., point-query IOPs) extend naturally to this case.

Exact proximity proofs. For tensor IOPs, we are often in the position where, provided that $\Delta(\mathbb{w}, R|_{(\mathbb{i}, \mathbb{x})}) > 0$, we have $\epsilon(\Delta(\mathbb{w}, R|_{(\mathbb{i}, \mathbb{x})})) = \epsilon$ where $\epsilon \in (0, 1)$ does not depend on the distance. In other words, whenever $(\mathbb{i}, \mathbb{x}) \notin L(R)$, then for any malicious prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept is at most ϵ. In such cases, we do not mention the distance function Δ, and use the terminology *exact interactive oracle proof of proximity* (exact IOPP), analogous to that used in [IW14] for probabilistically-checkable proofs.

4 Statement of main theorem

Our main theorem is a holographic IOP with point queries for rank-1 constraint satisfiability, a problem that is captured by the following definition.

Definition 4.1 (R1CS)**.** *The indexed relation R_{R1CS} is the set of all triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((A, B, C), (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}, n_{\mathrm{in}}, x), w\big)$$

where $\mathbb{F}$ is a finite field, A, B, C are matrices in $\mathbb{F}^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$, each having at most M non-zero entries, $x \in \mathbb{F}^{M, n_{\mathrm{row}}, n_{\mathrm{col}}, n_{\mathrm{in}}}$, $w \in \mathbb{F}^{n_{\mathrm{col}} - M, n_{\mathrm{row}}, n_{\mathrm{col}}, n_{\mathrm{in}}}$, and $z := (x, w) \in \mathbb{F}^{n_{\mathrm{col}}}$ is a vector such that $Az \circ Bz = Cz$. (Here "$\circ$" denotes the entry-wise product between two vectors.)

Below we denote by $N := \max(n_{\mathrm{row}}, n_{\mathrm{col}})$ the maximum of the number of constraints n_{row} and the number of variables n_{col}; and by E the number of operations in $\mathbb{F}$ required to check that z satisfies the R1CS relation by computing Az, Bz, and Cz and then checking that $Az \circ Bz = Cz$. Note that $E = O(M)$ always, but it can also be that E is much less than M (e.g., if the matrices A, B, C have some structure that enables computing Az, Bz, Cz from z in a way that is different than explicitly computing the matrix-vector products).

Theorem 4.2 (main)**.** *Let $\mathbb{F}$ be a finite field, and let $\lambda, t \in \mathbb{N}$ be parameters. There is a public-coin IOP (with point queries) for R_{R1CS} restricted to the field $\mathbb{F}$ that has the following efficiency:*

- *soundness error is $O(N/|\mathbb{F}|) + 2^{-\Omega_t(\lambda)}$;*
- *round complexity is $O(t + \log N)$;*
- *proof length is $O_t(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(\lambda t \cdot N^{1/t})$;*
- *the prover uses $O_t(E + N)$ field operations; and*
- *the verifier uses $O_t(E + \lambda N^{1/t})$ field operations.*

Moreover, the above public-coin IOP can be extended to the holographic setting, with the following changes:

- *soundness error increases to $O(N/|\mathbb{F}|) + 2^{-\Omega_t(\lambda)}$;*
- *an indexer uses $O_t(M)$ field operations to encode the index (i.e., the coefficient matrices) for the verifier;*
- *the prover arithmetic complexity increases to $O_t(M + N)$;*
- *the verifier arithmetic complexity decreases to $O_t(\lambda N^{1/t})$ (given oracle access to the encoded index).*

Our proof of the theorem has two parts.

- First, in Section 5, we construct an IOP with tensor queries for R_{R1CS}, with linear proof length and constant query complexity, where the prover and verifier have linear arithmetic complexity. For this we require the field to only have logarithmic size. Then in Section 6, we extend this protocol to additionally achieve a sublinear-time verifier in the holographic setting, at the cost requiring the field to have linear size.

- Second, in Section 8, we provide a transformation that converts any IOP with tensor queries into an IOP with point queries (and this transformation also supports the holographic setting). The main efficiency features are that the arithmetic complexity of the prover is preserved up to constants (preserving, in particular, linear complexity), and each tensor query is simulated with a small (sublinear) number of point queries. The transformation follows from two sub-protocols: a consistency test that we describe in Section 9, and a proximity test that we describe in Section 10.

Remark 4.3 (choice of field)**.** Unlike many other results about efficient probabilistic proofs, our result does *not* rely on any structure of the finite field $\mathbb{F}$ (e.g., smoothness of certain subgroups of $\mathbb{F}$). The only requirement on $\mathbb{F}$ is that it be large enough for the soundness error to be non-trivial.

5 A tensor IOP for constraint systems

We construct a tensor IOP for rank-1 constraint systems (see Definition 4.1), with linear proof length and constant query complexity, where the prover has *linear* arithmetic complexity.

For ease of exposition, we assume that R1CS instances have $n_{\mathrm{row}}, n_{\mathrm{col}} = k^t$ and $n_{\mathrm{in}} = k^{t_{\mathrm{in}}}$ for some $t_{\mathrm{in}} \in [t]$. This can always be achieved by padding with zero entries and does not affect our asymptotic results.

Theorem 5.1. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor IOP for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}$ with matrix dimensions $n_{\mathrm{row}}, n_{\mathrm{col}} = k^t$ and $n_{\mathrm{in}} = k^{t_{\mathrm{in}}}$ and has the following parameters:*
- *soundness error is $O(t \log k/|\mathbb{F}|)$;*
- *round complexity is $O(t \log k)$;*
- *proof length is $O(k^t)$ elements in $\mathbb{F}$;*
- *query complexity is $O(1)$;*
- *the prover and verifier both use $O(k^t + E)$ field operations.*

The statement "there exists w such that, for $z := (x, w)$, it holds that $Az \circ Bz = Cz$" is equivalent to the statement "there exist w, z_A, z_B, z_C such that, for $z := (x, w)$, it holds that $z_A = Az, z_B = Bz, z_C = Cz, z_A \circ z_B = z_C$". Satisfiability of the constraint system is thus reduced to four conditions of two types.

The protocol will start with the prover sending an oracle that contains a (claimed) satisfying assignment $z \in \mathbb{F}^{n_{\mathrm{col}}}$ and corresponding (claimed) linear combinations $z_A, z_B, z_C \in \mathbb{F}^{n_{\mathrm{row}}}$. In the rest of the protocol the prover will convince the verifier that the four conditions hold for these vectors.

The verifier replies with randomness that defines a challenge vector $r \in \mathbb{F}^{n_{\mathrm{row}}}$ (we discuss how it is sampled later). The conditions $z_A = Az, z_B = Bz, z_C = Cz$ are now replaced with the conditions $\langle r, z_A\rangle = \langle r^\intercal A, z\rangle, \langle r, z_B\rangle = \langle r^\intercal B, z\rangle, \langle r, z_C\rangle = \langle r^\intercal C, z\rangle$, which are equivalent up to a small soundness error over the choice of challenge vector r. Our choice of challenge vector r ensures that the verifier can directly obtain the inner products $\gamma_A := \langle r, z_A\rangle, \gamma_B := \langle r, z_B\rangle, \gamma_C := \langle r, z_C\rangle$ via tensor queries to z_A, z_B, z_C. Moreover, both the prover and verifier can locally compute the three vectors $r_A := r^\intercal A, r_B := r^\intercal B, r_C := r^\intercal C$, and then verifier can check the conditions $\gamma_A = \langle r_A, z\rangle, \gamma_B = \langle r_B, z\rangle, \gamma_C = \langle r_C, z\rangle$.

The prover and verifier engage in a *Hadamard product sub-protocol*, which we describe in Section 5.3, to check the condition "$z_A \circ z_B = z_C$". The prover and verifier also engage in three executions of a *scalar product sub-protocol*, which we describe in Section 5.2, to check the three conditions $\gamma_A = \langle r_A, z\rangle, \gamma_B = \langle r_B, z\rangle, \gamma_C = \langle r_C, z\rangle$. This leaves the verifier with one last task: checking that the full satisfying assignment z is consistent with the partial assignment x. The verifier does this by querying z at a random tensor that is padded with zeros at locations outside the partial assignment, and comparing the result with querying x with the same random tensor (this computation can be done locally).

We now proceed to a formal description of the protocol, and then prove its properties.

Construction 5.2 (tensor IOP for R1CS)**.** We construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{R1CS}. The prover $\mathbf{P}$ takes as input an index $\mathbb{i} = (A, B, C)$, instance $\mathbb{x} = (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}, n_{\mathrm{in}}, x)$, and witness $\mathbb{w} = w$, while the verifier $\mathbf{V}$ takes as input the index $\mathbb{i}$ and the instance $\mathbb{x}$.

- The prover $\mathbf{P}$ constructs the full assignment $z := (x, w) \in \mathbb{F}^{k^t}$, computes the three vectors $z_A := Az, z_B := Bz, z_C := Cz$ in $\mathbb{F}^{k^t}$, and sends the oracle message $\Pi_1 := (z, z_A, z_B, z_C) \in \mathbb{F}^{4 \cdot k^t}$.
- The verifier $\mathbf{V}$ sends uniformly random challenge vectors $r_1, \ldots, r_t \in \mathbb{F}^k$.

- The prover $\mathbf{P}$ derives the challenge vector $r = r_1 \otimes \cdots \otimes r_t$ in $\mathbb{F}^{k^t}$, computes the elements $\gamma_A := \langle r, z_A \rangle, \gamma_B := \langle r, z_B \rangle, \gamma_C := \langle r, z_C \rangle$, and sends the non-oracle message $(\gamma_A, \gamma_B, \gamma_C) \in \mathbb{F}^3$.
- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ compute the three vectors $r_A := r^\intercal A, r_B := r^\intercal B, r_C := r^\intercal C$ in $\mathbb{F}^{k^t}$, and then engage in several sub-protocols. Note that r_A, r_B, and r_C are known to $\mathbf{V}$, who will compute the answers to tensor queries on these vectors by itself.[14]
  - A Hadamard-product protocol with $\mathbb{w} = (z_A, z_B, z_C)$ and $\mathbb{x} = (\mathbb{F}, n_{\text{row}})$ to show that $z_A \circ z_B = z_C$.
  - A scalar-product protocol with $\mathbb{w} = (r_A, z)$ and $\mathbb{x} = (\mathbb{F}, n_{\text{col}}, \gamma_A)$ to show that $\langle r_A, z \rangle = \gamma_A$.
  - A scalar-product protocol with $\mathbb{w} = (r_B, z)$ and $\mathbb{x} = (\mathbb{F}, n_{\text{col}}, \gamma_B)$ to show that $\langle r_B, z \rangle = \gamma_B$.
  - A scalar-product protocol with $\mathbb{w} = (r_C, z)$ and $\mathbb{x} = (\mathbb{F}, n_{\text{col}}, \gamma_C)$ to show that $\langle r_C, z \rangle = \gamma_C$.
- The verifier $\mathbf{V}$ queries each of z_A, z_B, z_C in Π_1 at the tensor $r := r_1 \otimes \cdots \otimes r_t$ in order to obtain the answers $\langle r, z_A \rangle$, $\langle r, z_B \rangle$, and $\langle r, z_C \rangle$, and checks that $\gamma_A = \langle r, z_A \rangle$, $\gamma_B = \langle r, z_B \rangle$, and $\gamma_C = \langle r, z_C \rangle$.

  Moreover, $\mathbf{V}$ checks that the (claimed) satisfying assignment z is consistent with the partial assignment x as follows: sample uniformly random challenge vectors $s_1, \ldots, s_{t_{\text{in}}} \in \mathbb{F}^k$; set $s_{t_{\text{in}}+1}, \ldots, s_t \in \mathbb{F}^k$ to all equal the vector $(1, 0, \ldots, 0) \in \mathbb{F}^k$; query z in Π_1 at the tensor $s := s_1 \otimes \cdots \otimes s_t$ in order to obtain the answer $\langle s, z \rangle$; and check that $\langle s, z \rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}}, x \rangle$.

Lemma 5.3 (completeness)**.** *Construction 5.2 has perfect completeness.*

Proof. Let $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((A, B, C), (\mathbb{F}, M, n_{\text{row}}, n_{\text{col}}, n_{\text{in}}, x), w\big) \in R_{\text{R1CS}}$. Then $z := (x, w)$ satisfies $Az \circ Bz = Cz$. The prover $\mathbf{P}$ computes $z_A = Az$ and similarly for B and C, so the equation $z_A \circ z_B = z_C$ holds.

This means that the triple $((z_A, z_B, z_C), (\mathbb{F}, n_{\text{row}}), \perp)$ is in the Hadamard product indexed relation R_{HP}, and so the Hadamard product sub-protocol succeeds. Moreover, for every choice of $r = r_1 \otimes \cdots \otimes r_t$ it holds that $\langle r, z_A \rangle = \langle r, Az \rangle = \langle r^\intercal A, z \rangle = \langle r_A, z \rangle$, which means that the triple $((r_A, z), (\mathbb{F}, n_{\text{col}}, \langle r, z_A \rangle), \perp)$ is in the scalar product indexed relation R_{SP}; similarly for B and C. So the scalar product sub-protocols succeed.

Finally we discuss the consistency check between the full assignment $z = (x, w)$ and the partial assignment x. For every choice of $s_1, \ldots, s_{t_{\text{in}}}$ and setting $s_{t_{\text{in}}+1} = \cdots = s_t = (1, 0, \ldots, 0) \in \mathbb{F}^k$ it holds that $\langle s_1 \otimes \cdots \otimes s_t, z \rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}}, x \rangle$, which is the equation checked by the verifier. □

Lemma 5.4 (soundness)**.** *Let ϵ_{HP} and ϵ_{SP} be the soundness errors of the Hadamard product and scalar-product protocols. Construction 5.2 has soundness error*

$$\epsilon := \max\left\{\epsilon_{\text{HP}}, \epsilon_{\text{SP}} + \frac{t}{|\mathbb{F}|}, \frac{t_{\text{in}}}{|\mathbb{F}|}\right\} \ .$$

Proof. Let $(\mathbb{i}, \mathbb{x}) = \big((A, B, C), (\mathbb{F}, M, n_{\text{row}}, n_{\text{col}}, n_{\text{in}}, x)\big) \notin L(R)$ and fix a malicious prover. Let $\Pi_1 = (z, z_A, z_B, z_C)$ be the first message sent by the malicious prover. At least one of the following cases must hold.

- *The Hadamard product is incorrect: $z_A \circ z_B \neq z_C$.* By soundness of the Hadamard-product protocol, the verifier accepts with probability at most ϵ_{HP}.

[14] It is always possible to ensure that the verifier receives the correct answers to queries which concern witness and proof messages which are explicitly known to the verifier. Suppose, for example, that $\mathbf{V}$ makes a query $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_j \rangle$ for some $j \in \{0, 1, \ldots, i\}$ and vectors q_0 over $\mathbb{F}$ and $q_1, \ldots, q_t \in \mathbb{F}^k$. Let $\Pi_j = (\Pi_{j,1}, \Pi_{j,2}) \in \mathbb{F}^{\ell_{j,1} \cdot k^t} \times \mathbb{F}^{\ell_{j,2} \cdot k^t}$, where $\Pi_{j,1}$ is explicitly known to the verifier, who only has tensor-query access to $\Pi_{j,2}$, however. Let $q_0 = (q_{0,1}, q_{0,2}) \in \mathbb{F}^{\ell_{j,1}} \times \mathbb{F}^{\ell_{j,2}}$. The verifier may compute $\langle q_{0,1} \otimes q_1 \otimes \cdots \otimes q_t, \Pi_{j,1} \rangle$ for themselves, since they know $\Pi_{j,1}$, and can obtain $\langle q_{0,2} \otimes q_1 \otimes \cdots \otimes q_t, \Pi_{j,2} \rangle$ through a tensor query.

- *A linear combination is incorrect: $z_A \neq Az$ or $z_B \neq Bz$ or $z_C \neq Cz$.* Suppose that $z_A \neq Az$, without loss of generality. We apply the Schwartz–Zippel lemma to the non-zero polynomial $P(r_1, \ldots, r_t) := \langle r, z_A \rangle - \langle r^\intercal A, z \rangle$ of total degree t, concluding that $\Pr_r[\langle r, z_A \rangle = \langle r^\intercal A, z \rangle] \leq t/|\mathbb{F}|$. Suppose that $\langle r, z_A \rangle \neq \langle r^\intercal A, z \rangle$ and let $\gamma_A \in \mathbb{F}$ be the claimed value of the inner product sent by the malicious prover in the second message. Either $\langle r, z_A \rangle \neq \gamma_A$ or $\langle r^\intercal A, z \rangle \neq \gamma_A$ (or both). In the former case, the verifier rejects due to the check that $\gamma_A = \langle r, z_A \rangle$. In the latter case, by the soundness of the scalar-product protocol, the verifier accepts with probability at most ϵ_{SP}. By a union bound, the probability of accepting is at most $\epsilon_{\text{SP}} + \frac{t}{|\mathbb{F}|}$.
- *Inconsistency with the partial assignment: $z \neq (x, w)$ for some w.* We apply the Schwartz–Zippel lemma to the non-zero polynomial $P(\vec{s}_1, \ldots, \vec{s}_{t_{\text{in}}}) := \langle s_1 \otimes \cdots \otimes s_t, z \rangle - \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}}, x \rangle$ of total degree t_{in}, concluding that $\Pr_{\vec{s}_1, \ldots, \vec{s}_{t_{\text{in}}}}[\langle s_1 \otimes \cdots \otimes s_t, z \rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}}, x \rangle] \leq t_{\text{in}}/|\mathbb{F}|$. The acceptance probability is at most $t_{\text{in}}/|\mathbb{F}|$.

□

Lemma 5.5 (prover time)**.** *Let $\mathsf{tp}_{\text{HP}}(l)$ and $\mathsf{tp}_{\text{SP}}(l)$ be the arithmetic complexities of the prover in the Hadamard product and scalar-product protocols for vectors of length l. The prover in Construction 5.2 has arithmetic complexity $O(k^t + E + \mathsf{tp}_{\text{HP}}(n_{\text{row}}) + \mathsf{tp}_{\text{SP}}(n_{\text{col}}))$.*

Proof. For the first message the prover must compute the vectors z_A, z_B, z_C, which takes $O(E)$ field operations, by directly left-multiplying z by the matrices A, B, C respectively.

The prover must compute the challenge vector $r = r_1 \otimes \cdots \otimes r_t$. The cost of successively computing $r_1 \otimes r_2$ from r_1, and then $r_1 \otimes r_2 \otimes r_3$ from $r_1 \otimes r_2$, and so on is $k^2 + k^3 + \ldots + k^t = O(k^t)$ field operations.

The prover must compute $\gamma_A = \langle r, z_A \rangle$, $\gamma_B = \langle r, z_B \rangle$ and $\gamma_C = \langle r, z_C \rangle$. This requires $3n_{\text{row}}$ multiplications and $3n_{\text{row}}$ additions.

For the scalar-product protocols the prover must compute the vectors r_A, r_B, r_C, which is tantamount to computing $A^\intercal r, B^\intercal r, C^\intercal r$. By the Transposition Principle (Lemma A.2), left-multiplication by the transpose of a matrix $X \in \mathbb{F}^{n_{\text{row}} \times n_{\text{col}}}$ has the same cost as left-multiplication by the matrix X, plus $|n_{\text{row}} - n_{\text{col}}|$ additions (assuming that no rows or columns of X are all zeros). Then, computing the vectors r_A, r_B, r_C costs $O(E + |n_{\text{row}} - n_{\text{col}}|) = O(E)$ field operations.

The Hadamard-product protocol is invoked on vectors of size n_{row}, requiring $\mathsf{tp}_{\text{HP}}(n_{\text{row}})$ field operations. Each of the three scalar-product protocols are invoked on vectors of size n_{col}, requiring $\mathsf{tp}_{\text{HP}}(n_{\text{col}})$ field operations.

Summing up, the prover's computational costs are $O(E + k^t + \mathsf{tp}_{\text{HP}}(n_{\text{row}}) + \mathsf{tp}_{\text{SP}}(n_{\text{col}}))$ operations. □

Lemma 5.6 (verifier time)**.** *Let $\mathsf{tv}_{\text{HP}}(l)$ and $\mathsf{tv}_{\text{SP}}(l)$ be the arithmetic complexities of the verifier in the Hadamard product and scalar-product protocols for vectors of length l. The verifier in Construction 5.2 has arithmetic complexity $O(E + k^{t_{\text{in}}} + \mathsf{tv}_{\text{HP}}(n_{\text{row}}) + \mathsf{tv}_{\text{SP}}(n_{\text{col}}))$.*

Proof. The verifier must compute the vectors r_A, r_B, r_C, just as the prover does, which we have already argued takes $O(E)$ field operations. The Hadamard-product protocol is invoked on vectors of size n_{row}, requiring $\mathsf{tv}_{\text{HP}}(n_{\text{row}})$ field operations. Each of the three scalar-product protocols are invoked on vectors of size n_{col}, requiring $\mathsf{tv}_{\text{HP}}(n_{\text{col}})$ field operations. Further, within the scalar-product protocols, the verifier makes a query on z_A, z_B, and z_C. Since each of these vectors is explicitly known to the verifier, the verifier must compute the answer to each query for itself, at a cost of $O(n_{\text{col}})$ field operations. Finally, the verifier must compute $\langle \vec{s}_1 \otimes \ldots \otimes \vec{s}_{t_{\text{in}}}, x \rangle$. Using the method of computing each successive partial tensor product, the cost is $O(k^{t_{\text{in}}})$ multiplications.

Summing up, the verifier's computational costs are $O(E + k^{t_{\text{in}}} + \mathsf{tv}_{\text{HP}}(n_{\text{row}}) + \mathsf{tv}_{\text{SP}}(n_{\text{col}}))$ operations. □

5.1 Summing polynomials over subgroups

The heart of our protocol in Section 5.2 is a sumcheck protocol. In preparation for that section, we present a lemma relating sums of multivariate polynomials over multiplicative subgroups of finite fields. The lemma is a multivariate extension of [BCRSVW19, Lemma 5.5], which underlies the univariate sumcheck protocol.

Lemma 5.7. *Let H be a multiplicative subgroup of a finite field $\mathbb{F}$ and let $p(X_1, \ldots, X_l) \in \mathbb{F}[X_1, \ldots, X_l]$. If we denote by $p_{i_1,\ldots,i_l} \in \mathbb{F}$ the coefficient of $X_1^{i_1} \cdots X_l^{i_l}$ in the polynomial $p(X_1, \ldots, X_l)$, then*

$$\sum_{\vec{\omega} \in H^l} p(\vec{\omega}) = \left(\sum_{\vec{i} \equiv \vec{0} \bmod |H|} p_{\vec{i}} \right) \cdot |H|^l \ . \tag{3}$$

Proof. We prove this by induction. Let $l = 1$, and let $p(X_1) = \sum_{i_1} p_{i_1} X^{i_1}$. Note that, since H is cyclic (as it is a multiplicative subgroup of a finite field), for all $\omega_1 \in H$, we have

$$\sum_{\omega_1 \in H} \omega_1^{i_1} = \begin{cases} |H| & \text{if } \omega_1 = 1 \\ 0 & \text{otherwise} \end{cases} .$$

Hence, since $\omega_1^{i_1} = 1$ for $i_1 \equiv 0 \bmod |H|$,

$$\sum_{\omega_1 \in H} p(\omega_1) = \sum_{\omega_1 \in H} \sum_{i_1} p_{i_1} \omega_1^{i_1} = \sum_{i_1} p_{i_1} \sum_{\omega_1 \in H} \omega_1^{i_1} = \left(\sum_{i_1 \equiv 0 \bmod |H|} p_{i_1} \right) \cdot |H| \ . \tag{4}$$

Suppose that the result holds for some $l \in \mathbb{N}$. We prove that the result holds for $l + 1$ variables. Write $p(X_1, \ldots, X_{l+1}) = \sum_{i_{l+1}} p^{(i_{l+1})}(X_1, \ldots, X_l) X_{l+1}^{i_{l+1}}$, with coefficients related by $p_{i_1,\ldots,i_{l+1}} = p^{(i_{l+1})}_{i_1,\ldots,i_l}$. Then

$$\begin{aligned} \sum_{\vec{\omega} \in H^l} p(\vec{\omega}) &= \sum_{\omega_{l+1} \in H} \omega_{l+1}^{i_{l+1}} \cdot \left(\sum_{\omega_1,\ldots,\omega_l \in H} p^{(i_{l+1})}(\omega_1, \ldots, \omega_l) \right) \\ &= \sum_{\omega_{l+1} \in H} \omega_{l+1}^{i_{l+1}} \cdot \left(\sum_{i_1,\ldots,i_l \equiv 0 \bmod |H|} p^{(i_{l+1})}_{i_1,\ldots,i_l} \right) \cdot |H|^l \\ &= \left(\sum_{\vec{i} \equiv \vec{0} \bmod |H|} p_{\vec{i}} \right) \cdot |H|^{l+1} \ , \end{aligned}$$

where the second equality follows from the induction hypothesis, and the third from Equation (4). □

We also state, without proof, a corollary of Lemma 5.7 that allows us to change which coefficients of p appear in the sum.

Corollary 5.8. *Let H, $\mathbb{F}$, and p be as in Lemma 5.7. Then for every integer vector $\vec{j} = (j_1, \ldots, j_l) \in \mathbb{N}^l$, and writing $\vec{\omega}^{\vec{j}}$ for the product $\omega_1^{j_1} \cdots \omega_l^{j_l}$,*

$$\sum_{\vec{\omega} \in H^l} p(\vec{\omega}) \cdot \vec{\omega}^{\vec{j}} = \left(\sum_{\vec{i} + \vec{j} \equiv \vec{0} \bmod |H|} p_{\vec{i}} \right) \cdot |H|^l \ . \tag{5}$$

In the next section we will obtain a scalar-product protocol based on the sumcheck protocol, by applying Corollary 5.8 with $H = \{-1, 1\}$, $\vec{j} = (1, \ldots, 1)$, and p equal to the product of multilinear polynomials encoding two vectors a and b. In this case, p will be quadratic in each variable, and the only term contributing to the right hand side of Equation (5) will be $p_{1,\ldots,1}$, which will be the scalar product $\langle a, b\rangle$. (See Lemma 5.16.)

Remark 5.9. We state an analogue of Corollary 5.8 for the case of *additive* subgroups, which implies that our results also hold for fields of characteristic 2 even though we choose H with $|H| = 2$.

Let H be an additive subgroup of $\mathbb{F}$. Let $a_0 := \sum_{\omega \in H} \omega^{|H|-1}$ be the (formal) linear term of the subspace polynomial $\prod_{h \in H}(X - h)$. Then

$$\sum_{\vec{\omega} \in H^l} p(\vec{\omega}) \cdot \vec{\omega}^{\vec{j}} = \left(\sum_{\vec{i}+\vec{j} \equiv -\vec{1} \bmod |H|} p_{\vec{i}} \right) \cdot |a_0|^l \ . \tag{6}$$

This means that for a field $\mathbb{F}$ of characteristic 2, we can recover the results of the next section by choosing $H = \{0, 1\}$ and $\vec{j} = (0, \ldots, 0)$ and check Equation (6) using the sumcheck protocol.

5.2 Scalar product protocol

Definition 5.10. *The* **scalar product** *relation R_{SP} is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, n, \tau), (a, b))$ where $a, b \in \mathbb{F}^n$, $\tau \in \mathbb{F}$, and $\langle a, b\rangle = \tau$.*

We give a tensor IOP for R_{SP}. Note we only need to support instances with $n = k^t$ for Construction 5.2, but we will need support for $n = \ell \cdot k^t$ later in Section 6.

Theorem 5.11. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor exact IOPP for the indexed relation R_{SP} that supports instances over $\mathbb{F}$ with $n = \ell \cdot k^t$ and has the following parameters: soundness error is $O(\log n/|\mathbb{F}|)$; round complexity is $O(\log n)$; proof length is $O(n)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(n)$ field operations; and the verifier uses $O(\ell + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i} = (a, b)$ is the index itself.*

We will actually give a tensor IOP for a more general relation.

Definition 5.12. *The* **twisted scalar product** *relation R_{TSP} is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, n, y, \tau), (a, b))$ where $a, b, y \in \mathbb{F}^n$, $\tau \in \mathbb{F}$, and $\langle a \circ y, b\rangle = \tau$.*

Theorem 5.13. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor exact IOPP for the relation R_{TSP} that supports instances over $\mathbb{F}$ with $n = \ell \cdot k^t$ and $y = y_0 \otimes y_1 \otimes \ldots \otimes y_t$ for $y_0 \in \mathbb{F}^\ell$, $y_1, \ldots, y_t \in \mathbb{F}^k$ and has the following parameters: soundness error is $O(\log n/|\mathbb{F}|)$; round complexity is $O(\log n)$; proof length is $O(n)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(n)$ field operations; and the verifier uses $O(\ell + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i} = (a, b)$ is the index itself.*

Note that if we set $y := 1^n$ in the above theorem then we recover "standard" scalar products. The extra "twisted" functionality will be used for our Hadamard-product protocol in Section 5.3.

The rest of this section is dedicated to proving Theorem 5.13.

Definition 5.14. *Let $n \in \mathbb{N}$ be a power of 2, and set $l := \log n$. Let $v \in \mathbb{F}^n$, whose entries we index via binary strings $(i_1, \ldots, i_l) \in \{0, 1\}^l$. We define the following l-variate polynomial:*

$$p_v(X_1, \ldots, X_l) := \sum_{i_1, \ldots, i_l \in \{0,1\}} v_{i_1,\ldots,i_l} \cdot X_1^{i_1} \cdots X_l^{i_l} \ .$$

Definition 5.15. *Let $n, l \in \mathbb{N}$ and $v \in \mathbb{F}^n$ be as in Definition 5.14. We define $\mathrm{rv}(v) \in \mathbb{F}^n$ to be the vector whose $(i_1, \ldots, i_l)$-th entry is given by $v_{1-i_1,\ldots,1-i_l}$.*

Lemma 5.16. *Let $n, l \in \mathbb{N}$ be as in Definition 5.14. Given vectors $a, b \in \mathbb{F}^n$, we define the following polynomial:*

$$p(X_1, \ldots, X_l) := p_a(X_1, \ldots, X_l) \cdot p_{\mathrm{rv}(b)}(X_1, \ldots, X_l) \ .$$

Then, we have

$$\langle a, b \rangle = \mathrm{Coeff}_{X_1 \cdots X_l}(p) = 2^{-l} \sum_{\omega_1,\ldots,\omega_l \in \{-1,1\}} p(\omega_1, \ldots, \omega_l) \cdot \omega_1 \cdots \omega_l \ .$$

Proof. Corollary 5.8 implies that $\mathrm{Coeff}_{X_1 \cdots X_l}(p) = 2^{-l} \sum_{\omega_1,\ldots,\omega_l \in \{-1,1\}} p(\omega_1, \ldots, \omega_l) \cdot \omega_1 \cdots \omega_l$. Indeed, since p has degree 2 in each variable, the sum on the right-hand side of Equation (5) has one term. Note that all contributions to $\mathrm{Coeff}_{X_1 \cdots X_l}(p)$ arise from multiplying $a_{i_1,\ldots,i_l} X_1^{i_1} \cdots X_l^{i_l}$ in p_a and $b_{i_1,\ldots,i_l} X_1^{1-i_1} \cdots X_l^{1-i_l}$ in $p_{\mathrm{rv}(b)}$. Summing each contribution $a_{i_1,\ldots,i_l} b_{i_1,\ldots,i_l}$, we get that $\mathrm{Coeff}_{X_1 \cdots X_l}(p) = \langle a, b \rangle$. □

In our protocol, we apply the lemma above to the vectors $a \circ y$ and b instead of a and b, so that $\langle a, b \rangle$ is replaced by the twisted scalar product $\langle a \circ y, b \rangle$. The prover and verifier thus run the sumcheck protocol on the polynomial $p(X_1, \ldots, X_l) \cdot X_1 \cdots X_l$. As in [Tha13; XZZPS19], the prover is able to compute the messages for the sumcheck protocol using $O(n)$ arithmetic operations.

For completeness, we present the details of the sumcheck protocol that we use, framed as an IOP with tensor queries. The tensor queries will be used by the verifier on a and b to obtain the evaluation of $p(X_1, \ldots, X_l)$ required for the final check. The functions below help to describe the verifier's tensor queries.

Definition 5.17. *The functions $\mathrm{mp}, \mathrm{rmp} \colon \mathbb{F}^{\log k} \to \mathbb{F}^k$ are defined as follows:*

$$\mathrm{mp}(r_1, \ldots, r_{\log k}) := \bigotimes_{j=1}^{\log k} (1, r_j) \quad \textit{and} \quad \mathrm{rmp}(r_1, \ldots, r_{\log k}) := \bigotimes_{j=1}^{\log k} (r_j, 1) \ .$$

Construction 5.18. We construct an exact interactive oracle proof of proximity $(\mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{TSP}. There is no indexer $\mathbf{I}$ because R_{TSP} always has $\mathbb{i} = \bot$. The prover $\mathbf{P}$ takes as input an index $\mathbb{i} = \bot$, instance $\mathbb{x} = (\mathbb{F}, n, y, \tau)$, and witness $\mathbb{w} = (a, b)$; the verifier $\mathbf{V}$ has (tensor) query access to the witness $\mathbb{w}$ and takes as input the index $\mathbb{i}$ and the instance $\mathbb{x}$. Letting $p(X_1, \ldots, X_l) := p_{a \circ y}(X_1, \ldots, X_l) \cdot p_{\mathrm{rv}(b)}(X_1, \ldots, X_l)$, the protocol proceeds as follows.

- For each round $j \in [l]$:
  - The prover $\mathbf{P}$ computes the quadratic polynomial

$$q_j(X_j) := 2^{-(l-j)} \sum_{\omega_{j+1},\ldots,\omega_l \in \{-1,1\}} p(\rho_1, \ldots, \rho_{j-1}, X_j, \omega_{j+1}, \ldots, \omega_l) \cdot \omega_{j+1} \cdots \omega_l \tag{7}$$

    and sends the non-oracle message $(q_j(-1), q_j(0), q_j(1)) \in \mathbb{F}^3$ to the verifier.
  - The verifier $\mathbf{V}$ computes $q_{j-1}(\rho_{j-1})$ using interpolation and the evaluations $(q_j(-1), q_j(0), q_j(1))$. In the first round ($j = 1$), $\mathbf{V}$ checks that $\frac{1}{2} \sum_{\omega_1 \in \{-1,1\}} q_1(\omega_1) \cdot \omega_1 = \tau$; else in other rounds ($j > 1$), $\mathbf{V}$ checks that $\frac{1}{2} \sum_{\omega_j \in \{-1,1\}} q_j(\omega_j) \cdot \omega_j = q_{j-1}(\rho_{j-1})$.
  - The verifier $\mathbf{V}$ sends a random challenge $\rho_j \in \mathbb{F}$ to the prover $\mathbf{P}$.

- The verifier $\mathbf{V}$ uses tensor queries to check that $p(\rho_1, \ldots, \rho_l) = q_l(\rho_l)$. Namely, $\mathbf{V}$ computes the two tensor queries

$$\nu_1 := \Big(y_0 \circ \mathrm{mp}(\rho_1, \ldots, \rho_{\log \ell}), y_1 \circ \mathrm{mp}(\rho_{\log \ell + 1}, \ldots, \rho_{\log \ell k}), \ldots, y_t \circ \mathrm{mp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n}) \Big) \ , \tag{8}$$

$$\nu_2 := \Big(\mathrm{rmp}(\rho_1, \ldots, \rho_{\log \ell}), \mathrm{rmp}(\rho_{\log \ell + 1}, \ldots, \rho_{\log \ell k}), \ldots, \mathrm{rmp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n}) \Big) \ . \tag{9}$$

  Next $\mathbf{V}$ queries a, b at ν_1, ν_2 to get $\sigma_a := \langle \otimes \nu_1, a \rangle, \sigma_b := \langle \otimes \nu_2, b \rangle$; then $\mathbf{V}$ checks that $\sigma_a \cdot \sigma_b = q_l(\rho_l)$.

Lemma 5.19. *Construction 5.18 has perfect completeness.*

Proof. Suppose that $\langle a \circ y, b \rangle = \tau$. In each round $j \in [l]$, the honest prover computes and sends the evaluations $(q_j(-1), q_j(0), q_j(1)) \in \mathbb{F}^3$. By Lemma 5.16, we know that

$$\tau = 2^{-l} \sum_{\omega_1, \ldots, \omega_l \in \{-1,1\}} p(\omega_1, \ldots, \omega_l) \cdot \omega_1 \cdots \omega_l \ .$$

By construction, for each $j \in [l]$, we have

$$\begin{aligned}
&\frac{1}{2} \sum_{\omega_j \in \{-1,1\}} q_j(\omega_j) \cdot \omega_j \\
=& \frac{1}{2} \sum_{\omega_j \in \{-1,1\}} \omega_j \cdot 2^{-(l-j)} \sum_{\omega_{j+1}, \ldots, \omega_l \in \{-1,1\}} p(\rho_1, \ldots, \rho_{j-1}, \omega_j, \omega_{j+1}, \ldots, \omega_l) \cdot \omega_{j+1} \cdots \omega_l \\
=& 2^{-(l-j+1)} \sum_{\omega_j, \ldots, \omega_l \in \{-1,1\}} p(\rho_1, \ldots, \rho_{j-1}, \omega_j, \ldots, \omega_l) \cdot \omega_j \cdots \omega_l \\
=& \, q_{j-1}(\rho_{j-1}) \ . \quad \text{(Here } q_{j-1}(\rho_{j-1}) \text{ is replaced by } \tau \text{ when } j = 1.\text{)}
\end{aligned}$$

This means that the verifier's round checks all pass. Next, the final check performed by the verifier is $\langle \otimes \nu_1, a \rangle \cdot \langle \otimes \nu_2, b \rangle = q_l(\rho_l)$. Note that by construction, $q_l(\rho_l) = p(\rho_1, \ldots, \rho_l) = p_{a \circ y}(\rho_1, \ldots, \rho_l) \cdot p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_l)$. Hence to argue that the verifier accepts we are left to argue two equalities:

- $p_{a \circ y}(\rho_1, \ldots, \rho_l) = \langle \otimes \nu_1, a \rangle$. Indeed, from Equation (12) we obtain that $\otimes \nu_1 = \bigotimes_{i=0}^{t} y_i \circ \bigotimes_{j=1}^{\log n} (1, \rho_j)$. When the entries of $\otimes \nu_1$ are indexed by l-bit strings, the entry at index $(i_1, \ldots, i_l)$ is $y_{i_1 \ldots i_t} \rho_1^{i_1} \ldots \rho_l^{i_l}$ as the index simply indicates whether a 1 or a ρ term was chosen in a particular term of the tensor-product expression for $\otimes \nu_1$. Thus $\langle \otimes \nu_1, a \rangle = \sum_{i_1, \ldots, i_l} a_{i_1, \ldots, i_l} \cdot y_{i_1 \ldots i_t} \cdot \rho_1^{i_1} \cdots \rho_l^{i_l} = p_{a \circ y}(\rho_1, \ldots, \rho_l)$.

- $p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_l) = \langle \otimes \nu_2, b \rangle$. From Equation (13) we obtain that $\otimes \nu_2 = \bigotimes_{j=1}^{\log n} (\rho_j, 1)$. Therefore $\langle \otimes \nu_2, b \rangle = \sum_{i_1, \ldots, i_l} b_{i_1, \ldots, i_l} \rho_1^{1-i_1} \ldots \rho_l^{1-i_l} = p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_l)$.

□

Lemma 5.20. *Construction 5.18 has soundness error* $\epsilon_{\mathrm{SP}} = \frac{2 \log n}{|\mathbb{F}|}$.

Proof. Suppose that $\langle a \circ y, b \rangle \neq \tau$. Fix a malicious prover, which determines certain next-message functions

$$d_1, d_2(\rho_1), \ldots, d_l(\rho_1, \ldots, \rho_{l-1}) \ ,$$

$$e_1, e_2(\rho_1), \ldots, e_l(\rho_1, \ldots, \rho_{l-1}) \ ,$$
$$f_1, f_2(\rho_1), \ldots, f_l(\rho_1, \ldots, \rho_{l-1}) \ .$$

where $(d_j, e_j, f_j) \in \mathbb{F}^3$ was the non-oracle message sent during the j-th round by the malicious prover. For each $j \in [l]$, these next-message functions induce quadratic polynomials $\tilde{q}_j(X_j)$ via interpolation. The verifier accepts if:

- for $j = 1$, we have $\frac{1}{2}(d_1 - f_1) = \tau$;
- for each $j \in \{2, \ldots, l\}$, we have $\frac{1}{2}(d_j - f_j) = \tilde{q}_{j-1}(\rho_{j-1})$;
- $p_{a \circ y}(\rho_1, \ldots, \rho_l) \cdot p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_l) = \tilde{q}_l(\rho_l)$.

The values on the left-hand side of the last equation are obtained via the tensor queries ν_1 and ν_2 respectively.

Let E_j be the event that $\tilde{q}_j(X_j) = q_j(X_j)$, where $q_j(X_j)$ is defined by Equation (7). Let W be the event that the verifier accepts. We prove, by induction, that for each $j \in [l]$, we have

$$\Pr[W] \leq 2(l - j + 1)/|\mathbb{F}| + \Pr[W | E_j \wedge \cdots \wedge E_l] \ .$$

For the base case ($j = l$) observe that $\Pr[W] \leq \Pr[W|E_l] + \Pr[W|\bar{E}_l]$. If $\overline{E_l}$ occurs then the polynomial

$$P(X_l) := p_{a \circ y}(\rho_1, \ldots, \rho_{l-1}, X_l) \cdot p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_{l-1}, X_l) - \tilde{q}_l(X_l)$$

is a non-zero polynomial of degree at most 2, and if W occurs then the verifier has picked ρ_l for which $P(\rho_l) = 0$. Hence by the Schwartz–Zippel lemma we have $\Pr[W|\overline{E_l}] \leq 2/|\mathbb{F}|$.

Assume that the induction hypothesis holds for some $j \in \{2, \ldots, l\}$. We prove that it also holds for $j - 1$:

$$\begin{aligned}
\Pr[W] &\leq \frac{2(l - j + 1)}{|\mathbb{F}|} + \Pr[W | E_j \wedge \cdots \wedge E_l] \\
&\leq \frac{2(l - j + 1)}{|\mathbb{F}|} + \Pr[W | \overline{E_{j-1}} \wedge E_j \wedge \cdots \wedge E_l] + \Pr[W | E_{j-1} \wedge E_j \wedge \cdots \wedge E_l] \\
&\leq \frac{2(l - j + 1)}{|\mathbb{F}|} + \frac{2}{|\mathbb{F}|} + \Pr[W | E_{j-1} \wedge E_j \wedge \cdots \wedge E_l] \ .
\end{aligned}$$

The third line follows from the Schwartz–Zippel lemma.

Finally, note that $\Pr[W|E_1 \wedge \cdots \wedge E_l] = 0$. This is because if the malicious prover sends the correct polynomial in the first step, then $\frac{1}{2}(d_1 - f_1) = \frac{1}{2}\sum_{\omega_1 \in \{-1,1\}} q_1(\omega_1) = \langle a \circ y, b \rangle \neq \tau$ and the verifier's first check will fail.

We conclude that $\Pr[W] \leq 2l/|\mathbb{F}|$, as claimed. □

Lemma 5.21. *The prover in Construction 5.18 has arithmetic complexity $O(n)$.*

Proof. We describe how the honest prover, given vectors $a, b, y \in \mathbb{F}^n$ and challenges $\rho_1, \ldots, \rho_l \in \mathbb{F}$, can compute the coefficients of the quadratic polynomials $q_1(X_1), \ldots, q_l(X_l)$ in $O(n)$ operations. (The honest prover can then evaluate each of these polynomials at the locations $-1, 0, 1$, and send the evaluations.)

We proceed as follows. First, we recall the definitions of the polynomials $p(X_1, \ldots, X_l)$ and $q_j(X_j)$ for each $j \in [l]$. Next, we introduce notation for the coefficients of partial evaluations of the polynomials $p_{a \circ y}(X_1, \ldots, X_l)$ and $p_{\mathrm{rv}(b)}(X_1, \ldots, X_l)$. Next, we express each $q_j(X_j)$ in terms these coefficients. Finally, we give an algorithm that computes the coefficients of $q_1(X_1), \ldots, q_l(X_l)$ in $O(n)$ operations.

Recall that $p(X_1, \ldots, X_l) = p_{a \circ y}(X_1, \ldots, X_l) \cdot p_{\mathrm{rv}(b)}(X_1, \ldots, X_l)$ and

$$q_j(X_j) = 2^{-(l-j)} \sum_{\omega_{j+1}, \ldots, \omega_l \in \{-1,1\}} p(\rho_1, \ldots, \rho_{j-1}, X_j, \omega_{j+1}, \ldots, \omega_l) \cdot \omega_{j+1} \cdots \omega_l \ .$$

Next we discuss partial evaluations of $p_{a\circ y}$ and $p_{\mathrm{rv}(b)}$. In order to give explicit formulas for computing the polynomials $q_j(X_j)$, we define, for each $j \in \{0, 1, \ldots, l\}$, the coefficients $a^{(j)}_{i_{j+1},\ldots,i_l}, b^{(j)}_{i_{j+1},\ldots,i_l} \in \mathbb{F}$ as follows:

$$p_{a\circ y}(\rho_1, \ldots, \rho_j, X_{j+1}, \ldots, X_l) := \sum_{i_{j+1},\ldots,i_l \in \{0,1\}} a^{(j)}_{i_{j+1},\ldots,i_l} \cdot X_{j+1}^{i_{j+1}} \cdots X_l^{i_l} \ ,$$
$$p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_j, X_{j+1}, \ldots, X_l) := \sum_{i_{j+1},\ldots,i_l \in \{0,1\}} b^{(j)}_{i_{j+1},\ldots,i_l} \cdot X_{j+1}^{i_{j+1}} \cdots X_l^{i_l} \ .$$

Observe that for $j = 0$, these are just the coefficients of $a \circ y$ and $\mathrm{rv}(b)$. Further, note that $a^{(l)} = p_{a\circ y}(\rho_1, \ldots, \rho_l)$ and $b^{(l)} = p_{\mathrm{rv}(b)}(\rho_1, \ldots, \rho_l)$. The following recurrence relations hold between coefficients:

$$a^{(j)}_{i_{j+1},\ldots,i_l} = a^{(j-1)}_{0,i_{j+1},\ldots,i_l} + \rho_j \cdot a^{(j-1)}_{1,i_{j+1},\ldots,i_l} \ ,$$
$$b^{(j)}_{i_{j+1},\ldots,i_l} = b^{(j-1)}_{0,i_{j+1},\ldots,i_l} + \rho_j \cdot b^{(j-1)}_{1,i_{j+1},\ldots,i_l} \ .$$

Next, we express each polynomial $q_j(X_j)$ in terms of the coefficients $a^{(j-1)}_{i_j,\ldots,i_l}$ and $b^{(j-1)}_{i_j,\ldots,i_l}$. We start by considering $p(\rho_1, \ldots, \rho_l) = a^{(l)} \cdot b^{(l)}$. Using the recurrence relations, we have $a^{(l)} = a_0^{(l-1)} + \rho_l \cdot a_1^{(l-1)}$ and similarly for $b^{(l)}$. Expanding $a^{(l)} \cdot b^{(l)}$, we see that

$$p(\rho_1, \ldots, \rho_l) = q_l(\rho_l) = a_0^{(l-1)} \cdot b_1^{(l-1)} + \rho_l \cdot \left(a_0^{(l-1)} \cdot b_0^{(l-1)} + a_1^{(l-1)} \cdot b_1^{(l-1)}\right) + \rho_l^2 \cdot a_1^{(l-1)} \cdot b_1^{(l-1)} \ . \quad (10)$$

Since $q_{l-1}(\rho_{l-1}) = \frac{1}{2}\left(q_l(1) - q_l(-1)\right)$, which is the middle term of Equation (10), we have $q_{l-1}(\rho_{l-1}) = a_0^{(l-1)} \cdot b_0^{(l-1)} + a_1^{(l-1)} \cdot b_1^{(l-1)}$. Expanding using the recurrence relations, one finds that

$$\begin{aligned} & q_{l-1}(\rho_{l-1}) \\ &= \left(\textstyle\sum_{i_l} a^{(l-1)}_{i_l} \cdot b^{(l-1)}_{i_l}\right) \\ &= \left(\textstyle\sum_{i_l} a^{(l-2)}_{0,i_l} \cdot b^{(l-2)}_{1,i_l}\right) + \rho_{l-1} \cdot \left(\textstyle\sum_{i_{l-1},i_l} a^{(l-2)}_{i_{l-1},i_l} \cdot b^{(l-2)}_{i_{l-1},i_l}\right) + \rho_{l-1}^2 \cdot \left(\textstyle\sum_{i_l} a^{(l-2)}_{1,i_l} \cdot b^{(l-2)}_{0,i_l}\right) \ . \end{aligned}$$

Continuing, we see that for each $j \in [l]$, we can express $q_j(X_j)$ in terms of the coefficients $a^{(j-1)}_{i_j,\ldots,i_l}$ and $b^{(j-1)}_{i_j,\ldots,i_l}$:

$$\begin{aligned} q_j(\rho_j) &= \left(\textstyle\sum_{i_{j+1},\ldots,i_l} a^{(j)}_{i_{j+1},\ldots,i_l} \cdot b^{(j)}_{i_{j+1},\ldots,i_l}\right) \\ &= \left(\textstyle\sum_{i_{j+1},\ldots,i_l} a^{(j-1)}_{0,i_{j+1},\ldots,i_l} \cdot b^{(j-1)}_{1,i_{j+1},\ldots,i_l}\right) + \rho_{l-1} \cdot \left(\textstyle\sum_{i_j,\ldots,i_l} a^{(j-1)}_{i_j,\ldots,i_l} \cdot b^{(j-1)}_{i_j,\ldots,i_l}\right) \\ &\qquad + \rho_{l-1}^2 \cdot \left(\textstyle\sum_{i_{j+1},\ldots,i_l} a^{(j-1)}_{1,i_{j+1},\ldots,i_l} \cdot b^{(j-1)}_{0,i_{j+1},\ldots,i_l}\right) \ . \end{aligned} \quad (11)$$

We now give an algorithm that computes the coefficients of $q_1(X_1), \ldots, q_l(X_l)$ in $O(n)$ arithmetic operations. For $j = 0$, the prover can compute the coefficients $a^{(0)}_{i_1,\ldots,i_l}$ from a and y in n multiplications, and already knows the coefficients $b^{(0)}_{i_1,\ldots,i_l}$. Then, for each $j \in [l]$:

- The prover has the coefficients $a^{(j-1)}_{i_j,\ldots,i_l}$ and $a^{(j-1)}_{i_j,\ldots,i_l}$ for every $(i_j, \ldots, i_l) \in \{0,1\}^{l-(j-1)}$.

- The three sums in Equation (11) giving the coefficients of $q_j(X_j)$ contain $4 \cdot 2^{l-j}$ terms in total. Given the values of $a^{(j-1)}_{i_j,\ldots,i_l}$ and $b^{(j-1)}_{i_j,\ldots,i_l}$ for every $(i_j, \ldots, i_l) \in \{0,1\}^{l-(j-1)}$ we can compute all of the terms in $4 \cdot 2^{l-j}$ multiplications and add them together in $4 \cdot 2^{l-j}$ additions to find the coefficients of $q_j(X_j)$.

- On receiving ρ_j from $\mathbf{V}$, the prover computes the coefficients $a^{(j)}_{i_{j+1},\ldots,i_l}$ and $a^{(j)}_{i_{j+1},\ldots,i_l}$ for every $(i_{j+1}, \ldots, i_l) \in \{0,1\}^{l-j}$ via the recurrence relations. This requires 2^{l-j} multiplications and 2^{l-j} additions. The prover need not compute $a^{(l)}$ and $b^{(l)}$ for $j = l$.

This means that the total cost of computing the quadratic polynomials $q_1(X_1), \ldots, q_l(X_l)$ is the sum of a geometric series and is $O(2^l) = O(n)$ arithmetic operations. Each of the polynomials $q_1(X_1), \ldots, q_l(X_l)$ can be evaluated to find the necessary evaluation points to send to the verifier when required. □

Lemma 5.22. *The verifier in Construction 5.18 has arithmetic complexity $O(\ell + tk)$.*

Proof. The verifier must compute the query

$$\nu_2 = \mathrm{rmp}(\rho_1, \ldots, \rho_{\log \ell}), \mathrm{rmp}(\rho_{\log \ell+1}, \ldots, \rho_{\log \ell k}), \ldots, \mathrm{rmp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n})) \ .$$

Recall that $\mathrm{rmp}(\rho_1, \ldots, \rho_{\log k}) = \bigotimes_{j=1}^{\log k}(\rho_j, 1)$. The cost of computing successive vectors $(1, \rho_1) \otimes (1, \rho_2)$, $(1, \rho_1) \otimes (1, \rho_2) \otimes (1, \rho_3)$, and so on until $\bigotimes_{j=1}^{\log k}(\rho_j, 1)$, is $2 + 4 + \cdots + k/2 = k - 2$ multiplications. The total cost of computing ν_2 is then $O(tk)$ multiplications.

$$\nu_1 := \Big(y_0 \circ \mathrm{mp}(\rho_1, \ldots, \rho_{\log \ell}), y_1 \circ \mathrm{mp}(\rho_{\log \ell+1}, \ldots, \rho_{\log \ell k}), \ldots, y_t \circ \mathrm{mp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n})\Big) \ , \tag{12}$$

$$\nu_2 := \Big(\mathrm{rmp}(\rho_1, \ldots, \rho_{\log \ell}), \mathrm{rmp}(\rho_{\log \ell+1}, \ldots, \rho_{\log \ell k}), \ldots, \mathrm{rmp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n})\Big) \ . \tag{13}$$

The verifier must also compute the query

$$\nu_1 = (y_0 \circ \mathrm{mp}(\rho_1, \ldots, \rho_{\log \ell}), y_1 \circ \mathrm{mp}(\rho_{\log \ell+1}, \ldots, \rho_{\log \ell k}), \ldots, y_t \circ \mathrm{mp}(\rho_{\log n/k+1}, \ldots, \rho_{\log n})) \ .$$

Noting that $\mathrm{mp}(r_1, \ldots, r_l)$ has the same entries as $\mathrm{rmp}(r_1, \ldots, r_l)$ but in reverse order, it takes tk multiplications to compute ν_1 from ν_2.

The verifier must compute, for each $j \in [l]$, the evaluation $q_j(\rho_j)$. Each q_j is a quadratic polynomial specified using 3 evaluations (received from the prover). This gives a total of $O(l)$ operations over all rounds.

Finally, the verifier uses a single multiplication to check the expression $\sigma_a \cdot \sigma_b = q_l(\rho_l)$.

Summing up, and recalling that $l = \log n$ and $n = k^t$, the verifier has arithmetic complexity $O(tk)$. □

Remark 5.23. Our sumcheck-based Construction 5.18 takes inspiration from the linear-time scalar-product protocol of [BCGGHJ17] in the ILC model. This may be surprising because [BCGGHJ17] was *not* phrased as a sumcheck protocol. Below we explain how to modify our Construction 5.18 to expose this connection.[15]

1. Rescale $p_{a \circ y}$ and $p_{\mathrm{rv}(b)}$ by multiplying by $X_1^{-1} \cdots X_l^{-1}$. For each $j \in [l]$, we now have $q_j(X) = d_j^- X_j^{-1} + d_j^0 + d_j^+ X_j$.

[15] The modification even leads to a modest reduction in the communication complexity: from $3l$ field elements in our protocol to $2l$ field elements in its modification (plus a reduction in prover complexity).

2. In Construction 5.18, for each $j \in [l]$, the prover specifies the quadratic polynomial $q_j(X_j)$ by sending three *evaluations*. Modify the prover to instead send the *coefficients* (d_j^-, d_j^0, d_j^+) of $q_j(X_j)$.

3. One can verify that $d_j^0 = \frac{1}{2}\sum_{\omega_j \in \{-1,1\}} q_j(\omega_j)\cdot\omega_j = q_{j-1}(\rho_{j-1}) = d_{j-1}^-\rho_{j-1}^{-1} + d_{j-1}^0 + d_{j-1}^+\rho_{j-1}$. When $j = 1$, we have $d_1^0 = \tau$. This enables us to *collapse* all l verification equations into one:

$$p(\rho_1, \ldots, \rho_l) = \tau + d_1^-\rho_1^{-1} + d_1^+\rho_1 + \cdot + d_l^-\rho_l^{-1} + d_l^+\rho_l \ .$$

The soundness analysis becomes reminiscent of the extension of the Schwartz–Zippel Lemma stated in [BCGGHJ17, Lemma 1].

The above modifications yield a protocol that closely resembles the ILC protocol for scalar products in [BCGGHJ17] when the ILC vector length is set to 1, and parameters are set to achieve a logarithmic number of rounds. With a little more effort, one can arrive at the vectorized reduced-round protocol of [BCGGHJ17].

5.3 Hadamard-product protocol

Definition 5.24. *The* **Hadamard product** *indexed relation R_{HP} is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, m), (a, b, c))$ where $a, b, c \in \mathbb{F}^m$ and $a \circ b = c$.*

We give a tensor IOP for R_{HP}. Note that as in Section 5.2, we only need to support instances with $m = k^t$ for Construction 5.2, but we will need support for $m = \ell \cdot k^t$ later in Section 6.

Theorem 5.25. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor exact IOPP for the indexed relation R_{HP} that supports instances over $\mathbb{F}$ with $m = \ell \cdot k^t$ and has the following parameters: soundness error is $O(\log m/|\mathbb{F}|)$; round complexity is $O(\log m)$; proof length is $O(m)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(m)$ field operations; and the verifier uses $O(\ell + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i} = (a, b, c)$ is the index itself.*

The prover wants to convince the verifier that $a \circ b = c$. The verifier begins the protocol by sending randomness that defines a challenge vector $y \in \mathbb{F}^m$. The condition $a \circ b = c$ is now replaced by the condition $\langle a \circ y, b\rangle = \langle c, y\rangle$, which is equivalent up to a small soundness error over the choice of y. Then we have an instance of R_{TSP} with $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((a, b), (\mathbb{F}, m, y, \langle c, y\rangle), \perp)$. The verifier can obtain $\langle c, y\rangle$ via a query.

Construction 5.26. We construct an exact interactive oracle proof of proximity $(\mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{HP}. There is no indexer $\mathbf{I}$ because R_{EP} always has $\mathbb{i} = \perp$. The prover $\mathbf{P}$ takes as input an index $\mathbb{i} = \perp$, instance $\mathbb{x} = (\mathbb{F}, m)$, and witness $\mathbb{w} = (a, b, c)$; the verifier has query access to the witness $\mathbb{w}$ and takes as input the index $\mathbb{i}$ and the instance $\mathbb{x}$. The protocol proceeds as follows.

- The verifier $\mathbf{V}$ sends uniformly random challenge vectors $y_0 \in \mathbb{F}^\ell$, $y_1, \ldots, y_t \in \mathbb{F}^k$.
- The prover $\mathbf{P}$ derives the challenge vector $y = y_0 \otimes y_1 \otimes \ldots \otimes y_t \in \mathbb{F}^{\ell \cdot k^t}$, computes the element $\tau_c = \langle c, y\rangle$, and sends the non-oracle message $\tau_c \in \mathbb{F}$.
- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in a twisted scalar-product protocol with $\mathbb{i} := (a, b)$ and $\mathbb{x} := (\mathbb{F}, m, (y_0, y_1, \ldots, y_t), \tau_c)$ to show that $\langle a \circ y, b\rangle = \tau_c$.
- The verifier $\mathbf{V}$ queries c at the tensor $y = y_0 \otimes y_1 \otimes \ldots \otimes y_t$ to obtain answer $\langle c, y\rangle$, and checks that $\tau_c = \langle c, y\rangle$.

Lemma 5.27. *Construction 5.26 has perfect completeness.*

Proof. Let $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, m), (a, b, c)) \in R_{\text{HP}}$. Then $a \circ b = c$. Take the scalar product of both sides of the expression with y, and $\langle y, a \circ b\rangle = \langle c, y\rangle$. Now, observe that $\langle y, a \circ b\rangle = \langle a \circ y, b\rangle$. Also, $\tau_c = \langle c, y\rangle$, which means that $\langle a \circ y, b\rangle = \tau_c$. Hence $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, m, (y_1, \ldots, y_t), \tau), (a, b)) \in R_{\text{TSP}}$ and the twisted scalar product sub-protocol succeeds. □

Lemma 5.28. *Construction 5.26 has soundness error* $\epsilon_{\text{HP}} = \epsilon_{\text{TSP}} + (t + 1)/|\mathbb{F}|$.

Proof. Suppose that $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, m), (a, b, c), \ \big) \notin L(R_{\text{HP}})$ and fix a malicious prover. Then $a \circ b \neq c$. With $y = y_0 \otimes y_1 \otimes \ldots \otimes y_t$, we apply the Schwartz–Zippel lemma to the non-zero polynomial $P(y_0, y_1, \ldots, y_t) := \langle a \circ y, b\rangle - \langle c, y\rangle$ of total degree t, concluding that $\Pr_y[\langle a \circ y, b\rangle = \langle c, y\rangle] \leq t/|\mathbb{F}|$. Suppose that $\langle a \circ y, b\rangle \neq \langle c, y\rangle$ and let $\tau_c \in \mathbb{F}$ be the claimed value of the inner product sent by the malicious prover in the second message. Either $\langle c, y\rangle \neq \tau_c$ or $\langle a \circ y, b\rangle \neq \tau_c$ (or both). In the former case, the verifier will reject due to the check that $\tau_c = \langle c, y\rangle$. In the latter case, by the soundness of the twisted scalar-product protocol, the verifier accepts with probability at most ϵ_{SP}. By a union bound, the probability of accepting is at most $\epsilon_{\text{TSP}} + t/|\mathbb{F}|$. □

Lemma 5.29. *Let* $\mathsf{tp}_{\text{TSP}}(l)$ *be the arithmetic complexity of the prover in the twisted scalar-product protocol for vectors of length* l*. The prover in Construction 5.26 has arithmetic complexity* $O(\ell k^t + \mathsf{tp}_{\text{TSP}}(m))$.

Proof. The prover must compute the challenge vector $y = y_0 \otimes y_1 \otimes \ldots \otimes y_t$. The cost of successively computing $y_0 \otimes y_1$, $y_0 \otimes y_1 \otimes y_2$, and then $y_0 \otimes y_1 \otimes y_2 \otimes y_3$, and so on, is $\ell k + \ell k^2 + \ell k^3 + \ldots + \ell k^t = O(\ell k^t)$ field operations. The prover must compute $\tau_c = \langle c, y\rangle$ which requires n multiplications and n additions. The twisted scalar-product protocol is invoked on vectors of size m, requiring $\mathsf{tp}_{\text{TSP}}(m)$ field operations.

Summing up, the prover's computational costs are $O(k^t + \mathsf{tp}_{\text{TSP}}(m))$ operations. □

Lemma 5.30. *Let* $\mathsf{tv}_{\text{TSP}}(l)$ *be the arithmetic complexities of the verifier in the Hadamard product and scalar-product protocols for vectors of length* l*. The verifier in Construction 5.26 has arithmetic complexity* $\mathsf{tv}_{\text{SP}}(m)$.

Proof. The twisted scalar-product protocol is invoked on vectors of size m, requiring $\mathsf{tv}_{\text{TSP}}(m)$ field operations. The verifier obtains $\langle c, y\rangle$ using a query, so the only other verification cost is an equality check. □

6 Achieving holography

We extend the tensor IOP for R1CS from Section 5 to additionally achieve holography, provided that the R1CS instance is defined over a large-enough (linear-size) field.

Theorem 6.1. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor holographic IOP for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}$ with $M = \ell \cdot k^t$, $n_{\mathrm{row}}, n_{\mathrm{col}} = k^t$ and $n_{\mathrm{in}} = k^{t_{\mathrm{in}}}$ and has the following parameters:*
- *soundness error is $O((M + k^t)/|\mathbb{F}|)$;*
- *round complexity is $O(\log(M + k^t))$;*
- *proof length is $O(M + k^t)$ elements in $\mathbb{F}$;*
- *query complexity is $O(1)$;*
- *the indexer uses $O(M)$ field operations;*
- *the prover uses $O(k^t + M)$ field operations;*
- *the verifier uses $O(k^{t_{\mathrm{in}}} + \ell + tk)$ field operations.*

We now explain how to modify Construction 5.2 into a new protocol (see Construction 6.4 below) that fulfills Theorem 6.1. To avoid the verifier having to multiply vectors by matrices, we replace scalar product protocols and other tests for checking that $z_A = Az$, $z_B = Bz$, and $z_C = Cz$, with subprotocols for the lincheck problem defined below. As explained in Section 2.5, making our construction holographic by simply allowing tensor query access to A, B, C written in dense form precludes running in linear time. Therefore, we define the lincheck problem to be relative to the natural sparse representation of a matrix U (a list of the values of its non-zero entries and their locations).

Definition 6.2. *Let $U \in \mathbb{F}^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$ be a matrix with M non-zero entries. The* **sparse representation** *of U consists of $\mathrm{val}_U \in \mathbb{F}^M$, $\mathrm{row}_U \in [n_{\mathrm{row}}]^M$ and $\mathrm{col}_U \in [n_{\mathrm{col}}]^M$ such that val_U contains the M non-zero entries of $U \in \mathbb{F}^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$ and, for all $\kappa \in [M]$, $\mathrm{val}_U(\kappa) = U(\mathrm{row}_U(\kappa), \mathrm{col}_U(\kappa))$.*

Definition 6.3. *The* **lincheck** *indexed relation R_{LC} is the set of all triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big(U, (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}), (v, v_{\mathrm{U}})\big)$$

where $U \in \mathbb{F}^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$ is a matrix with M non-zero entries given in sparse representation, $v \in \mathbb{F}^{n_{\mathrm{col}}}$, $v_{\mathrm{U}} \in \mathbb{F}^{n_{\mathrm{row}}}$, $\mathrm{val}_U \in \mathbb{F}^M$, $\mathrm{row}_U \in [n_{\mathrm{row}}]^M$, $\mathrm{col}_U \in [n_{\mathrm{col}}]^M$ and $Uv = v_{\mathrm{U}}$.

Construction 6.7 gives a protocol for R_{LC}.

In the holographic protocol for R_{R1CS} which achieves Theorem 6.1, the prover and verifier use the protocol of Construction 6.7 three times, with

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big(U, (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}), (z, z_U)\big)$$

for $U \in \{A, B, C\}$.

Construction 6.4 (tensor IOP for R1CS)**.** We construct a holographic interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{R1CS}. Given as input an index $\mathbb{i} = (A, B, C)$, the indexer $\mathbf{I}$ runs the indexer for the lincheck protocol (Construction 6.7) on input U to obtain $(\mathrm{row}_U, \mathrm{col}_U, [n_{\mathrm{row}}], [n_{\mathrm{col}}])$ for each $U \in \{A, B, C\}$, and outputs the oracle message $\Pi_0 := (A, B, C, [n_{\mathrm{row}}], [n_{\mathrm{col}}])$. The prover $\mathbf{P}$ takes as input the index $\mathbb{i}$, instance $\mathbb{x} = (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}, n_{\mathrm{in}}, x)$, and witness $\mathbb{w} = w$, while the verifier $\mathbf{V}$ has query access to the index $\mathbb{i}$ and takes as input the instance $\mathbb{x}$.

- The prover $\mathbf{P}$ constructs the full assignment $z := (x, w) \in \mathbb{F}^{k^t}$, computes the three vectors $z_A := Az, z_B := Bz, z_C := Cz$ in $\mathbb{F}^{k^t}$, and sends the oracle message $\Pi_1 := (z, z_A, z_B, z_C) \in \mathbb{F}^{4 \cdot k^t}$.
- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in several sub-protocols.
  - A Hadamard-product protocol with $\mathbb{x} = (\mathbb{F}, n_{\text{row}})$ and $\mathbb{w} = (z_A, z_B, z_C)$ to show that $z_A \circ z_B = z_C$.
  - A lincheck protocol with $\mathbb{i} = A$, $\mathbb{x} = (\mathbb{F}, M, n_{\text{row}}, n_{\text{col}})$, and $\mathbb{w} = (z, z_A)$ to show that $z_A = Az$.
  - A lincheck protocol with $\mathbb{i} = B$, $\mathbb{x} = (\mathbb{F}, M, n_{\text{row}}, n_{\text{col}})$, and $\mathbb{w} = (z, z_B)$ to show that $z_B = Bz$.
  - A lincheck protocol with $\mathbb{i} = C$, $\mathbb{x} = (\mathbb{F}, M, n_{\text{row}}, n_{\text{col}})$, and $\mathbb{w} = (z, z_C)$ to show that $z_C = Cz$.
- The verifier $\mathbf{V}$ checks that the (claimed) satisfying assignment z is consistent with the partial assignment x as follows: sample uniformly random challenge vectors $s_1, \ldots, s_{t_{\text{in}}} \in \mathbb{F}^k$; set $s_{t_{\text{in}}+1}, \ldots, s_t \in \mathbb{F}^k$ to all equal the vector $(1, 0, \ldots, 0) \in \mathbb{F}^k$; query z in Π_1 at the tensor $s := s_1 \otimes \cdots \otimes s_t$ in order to obtain the answer $\langle s, z \rangle$; and check that $\langle s, z \rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}}, x \rangle$.

The rest of this section is dedicated to describing the construction of a holographic lincheck protocol and its subprotocols.

6.1 Holographic lincheck

We give a holographic lincheck protocol, which supports instances of R_{LC} with $M = \ell \cdot k^t$ and $n_{\text{row}}, n_{\text{col}} = k^t$.

Lemma 6.5. *For every finite field $\mathbb{F}$ and positive integers k, t, Construction 6.7 is an exact $(\mathbb{F}, k, t)$-tensor holographic IOPP for the indexed relation R_{LC} that supports instances over $\mathbb{F}$ with $M = \ell \cdot k^t$, $n_{\text{row}}, n_{\text{col}} = k^t$ and has the following parameters: soundness error is $O((M+k^t)/|\mathbb{F}|)$; round complexity is $O(\log(M+k^t))$; proof length is $O(M + k^t)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(M + k^t)$ field operations; and the verifier uses $O(M/k^t + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i}$ is the index itself.*

Our construction follows the strategy outlined in Section 2.5. For a random challenge vector $r = r_1 \otimes \cdots \otimes r_t$ the verifier checks whether $r^\intercal v_{\text{U}} = r^\intercal U v$, obtaining the left-hand side directly by using r as a tensor query on v_{U}. The verifier obtains the right-hand side directly from the prover and then interacts with the prover to check that it was computed correctly according to the following expression:

$$r^\intercal U v = \sum_{\kappa \in [M]} U(\text{row}_U(\kappa), \text{col}_U(\kappa)) \cdot r(\text{row}_U(\kappa)) \cdot v(\text{col}_U(\kappa)) = \sum_{\kappa \in [M]} \text{val}_U(\kappa) \cdot r(\text{row}_U(\kappa)) \cdot v(\text{col}_U(\kappa)) \ .$$

Towards this, the prover sends r, $r^* := (r(\text{row}_U(\kappa)))_{\kappa \in [M]}$, and $v^* := (v(\text{col}_U(\kappa)))_{\kappa \in [M]}$. The prover convinces the verifier that the right-hand side was computed from val_U, r^*, v^* by using Hadamard product and scalar product protocols and, moreover, convinces the verifier that the entries of r^* and v^* were correctly selected from the entries of r and v. The latter is done via two look-up protocols for the following relation.

Definition 6.6. *The* **look-up** *indexed relation R_{LU} is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((I, N), (\mathbb{F}, M, N), (c, d))$ where $|I| = M$, $c \in \mathbb{F}^M$, $d \in \mathbb{F}^N$, and $\{(c_i, I_i)\}_{i \in [M]} \subseteq \{(d_j, j)\}_{j \in [N]}$.*

Setting $(c, I) := (v^*, \text{col}_U)$ and $(d, N) := (v, k^t)$ demonstrates that all entries of the prover's message v^* correspond correctly to the entries of v. Similarly set $(c, I) := (r^*, \text{row}_U)$ and $(d, N) := (r, k^t)$ to prove c is correct.

Finally, the prover must convince the verifier that r is consistent with $r_1, \ldots, r_t$. This is done using the tensor-consistency test described in Section 6.2.

Construction 6.7. We construct an exact holographic interactive oracle proof of proximity $\mathsf{IOPP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{LC}. Given as input an index $\mathtt{i} = U$, the indexer $\mathbf{I}$ runs the indexer for the look-up protocol (Construction 6.17) on inputs $(\mathrm{row}_U, n_{\mathrm{row}})$ and $(\mathrm{col}_U, n_{\mathrm{col}})$ to obtain $(\mathrm{row}_U, [n_{\mathrm{row}}])$ and $(\mathrm{col}_U, [n_{\mathrm{col}}])$ and outputs the oracle message $\Pi_0 := (\mathrm{row}_U, \mathrm{col}_U, \mathrm{val}_U, [n_{\mathrm{row}}], [n_{\mathrm{col}}])$. The prover $\mathbf{P}$ takes as input the index $\mathtt{i}$, the instance $\mathtt{x} = (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}})$, and the witness $(\mathtt{w} = v, v_{\mathrm{U}})$; the verifier $\mathbf{V}$ has query access to the index $\mathtt{i}$ and the witness $\mathtt{w}$ and takes as input instance $\mathtt{x}$.

- The prover constructs $v^* = ((v)_{\mathrm{col}_U(\kappa)})_{\kappa \in [M]} \in \mathbb{F}^M$. The prover sends the oracle message $v^* \in \mathbb{F}^{k^t}$.
- The verifier $\mathbf{V}$ sends uniformly random challenge vectors $r_1, \ldots, r_t \in \mathbb{F}^k$.
- The prover derives the challenge vector $r = r_1 \otimes \cdots \otimes r_t \in \mathbb{F}^{k^t}$. The prover constructs $r^* := ((r)_{\mathrm{row}_U(\kappa)})_{\kappa \in [M]}$. The prover computes $e = r^* \circ v^*$. The prover computes $\gamma = \langle r, v_{\mathrm{U}} \rangle$. The prover sends the oracle message $(r, r^*, e) \in \mathbb{F}^{k^t + 2 \cdot M}$. The prover sends the non-oracle message $\gamma \in \mathbb{F}$.
- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in several sub-protocols.
  - A look-up protocol with $\mathtt{i} = (\mathrm{row}_U, n_{\mathrm{row}})$, $\mathtt{x} = (\mathbb{F}, M, n_{\mathrm{row}})$ and $\mathtt{w} = (r^*, r)$ to show that $(r^*, \mathrm{row}_U) \subseteq (r, [n_{\mathrm{row}}])$.
  - A look-up protocol with $\mathtt{i} = (\mathrm{col}_U, n_{\mathrm{col}})$, $\mathtt{x} = (\mathbb{F}, M, n_{\mathrm{col}})$ and $\mathtt{w} = (v^*, v)$ to show that $(v^*, \mathrm{col}_U) \subseteq (v, [n_{\mathrm{col}}])$.
  - A Hadamard-product protocol with $\mathtt{x} = (\mathbb{F}, M)$ and $\mathtt{w} = (e, r^*, v^*)$ to show that $e = r^* \circ v^*$.
  - A scalar-product protocol with $\mathtt{x} = (\mathbb{F}, M, \gamma)$ and $\mathtt{w} = (\mathrm{val}_U, e)$ to show that $\langle \mathrm{val}_U, e \rangle = \gamma$.
- The verifier $\mathbf{V}$ queries v_{U} in Π_1 at the tensor $r_1 \otimes \cdots \otimes r_t$ in order to obtain the answer $\langle r_1 \otimes \cdots \otimes r_t, v_{\mathrm{U}} \rangle$ and checks that $\gamma = \langle r_1 \otimes \cdots \otimes r_t, v_{\mathrm{U}} \rangle$.

  Moreover, $\mathbf{V}$ checks that the (oracle message) r is consistent with the tensor product $r_1 \otimes \cdots \otimes r_t$ using a single query and the tensor-consistency test of Claim 6.9.

Properties of Construction 6.7. Below in Lemma 6.8 we prove that Construction 6.7 has the soundness error stated in Lemma 6.5. Before that, we informally discuss (but omit formal proofs of) the other efficiency parameters claimed in Lemma 6.5. The prover runs a look-up protocol with instance $\mathtt{x} = (\mathbb{F}, M, k^t)$. This dominates the contributions to all efficiency parameters, so all values in Lemma 6.5 are inherited from Lemma 6.14, where the length of the look-up table is k^t and the length of the vector to be looked-up is M.

Lemma 6.8. *Construction 6.7 has soundness error* $\epsilon_{\mathrm{LC}} = \frac{t}{|\mathbb{F}|} + \max\{\epsilon_{\mathrm{LU}}(n_{\mathrm{row}}), \epsilon_{\mathrm{LU}}(n_{\mathrm{col}}), \epsilon_{\mathrm{HP}}(M), \epsilon_{\mathrm{SP}}(M), \frac{t}{|\mathbb{F}|}\}$.

Proof. Let $(\mathtt{i}, \mathtt{x}, \mathtt{w}) = \big((U, n_{\mathrm{row}}, n_{\mathrm{col}}), (\mathbb{F}, M, n_{\mathrm{row}}, n_{\mathrm{col}}), v, v_{\mathrm{U}}\big) \notin L(R_{\mathrm{LC}})$. This means that $v_{\mathrm{U}} \neq Uv$.

Fix a malicious prover, which determines certain next-message functions for oracle messages v^*, $r(r_1, \ldots, r_t)$, $r^*(r_1, \ldots, r_t)$, $e(r_1, \ldots, r_t)$; and non-oracle message $\gamma(r_1, \ldots, r_t)$.

We apply the Schwartz–Zippel lemma to the non-zero polynomial $P(r_1, \ldots, r_t) := \langle r, v_{\mathrm{U}} \rangle - \langle r^{\intercal} U, v \rangle$ of total degree t, concluding that except with probability at most $t/|\mathbb{F}|$, we have $\langle r, v_{\mathrm{U}} \rangle \neq \langle r^{\intercal} U, v \rangle$. This means that least one of the following cases must hold, and the soundness error follows by a union bound.

- $(r^*, \mathrm{row}_U) \not\subseteq (r, [n_{\mathrm{row}}])$ *or* $(v^*, \mathrm{col}_U) \not\subseteq (v, [n_{\mathrm{col}}])$. By soundness of the look-up protocol, the verifier accepts with probability at most $\epsilon_{\mathrm{LU}}(n_{\mathrm{row}})$ or $\epsilon_{\mathrm{LU}}(n_{\mathrm{col}})$.
- *The Hadamard product is incorrect:* $e \neq r^* \circ v^*$. By the soundness of the Hadamard-product protocol, the verifier accepts with probability at most $\epsilon_{\mathrm{HP}}(M)$.
- *The scalar product is incorrect:* $\langle \mathrm{val}_U, e \rangle \neq \gamma$. If $\langle \mathrm{val}_U, e \rangle \neq \gamma$ then by soundness of the scalar-product protocol, the verifier accepts with probability at most $\epsilon_{\mathrm{SP}}(M)$.

- *The query answer is incorrect:* $\langle r_1 \otimes \cdots \otimes r_t, v_{\mathrm{U}} \rangle \neq \gamma$. In this case, the verifier rejects because they check this directly.
- $r \neq r_1 \otimes \cdots \otimes r_t$. By soundness of the tensor consistency check (see Claim 6.9), the verifier accepts with probability at most $t/|\mathbb{F}|$.

□

The rest of this section is devoted to constructing a look-up protocol, which requires several subprotocols.

6.2 Tensor-consistency test

In Construction 6.7, $\mathbf{V}$ knows $r_1, \ldots, r_t \in \mathbb{F}^k$ and must check that $\mathbf{P}$ has sent $r \in \mathbb{F}^{k^t}$ that equals the tensor $r_1 \otimes \cdots \otimes r_t$. If so, we know that $r(i_1, \ldots, i_t) = r_1(i_1) \otimes \cdots \otimes r_t(i_t)$ for every tuple $(i_1, \ldots, i_t) \in [k]^t$. In particular, we also know that, for every tensor $s = s_1 \otimes \cdots \otimes s_t \in \mathbb{F}^{k^t}$, it holds that

$$\langle r, s \rangle = \langle r_1, s_1 \rangle \cdots \langle r_t, s_t \rangle \ . \tag{14}$$

Conversely, if r does not equal the tensor $r_1 \otimes \cdots \otimes r_t$, then by the Schwartz–Zippel lemma the probability that Equation (14) holds for a random tensor s is at most $t/|\mathbb{F}|$.

This means that for the verifier it suffices to check the equation for a random tensor s, as we follows. The verifier samples random $s_1, \ldots, s_t \in \mathbb{F}^k$, and queries r at the tensor $s = s_1 \otimes \cdots \otimes s_t \in \mathbb{F}^{k^t}$ to obtain the left-hand side of Equation (14). The verifier can directly compute the right-hand side of Equation (14) in $O(tk)$ arithmetic operations, as each $\langle r_j, s_j \rangle$ is a scalar product of vectors of length k and so can be evaluated in $O(k)$ arithmetic operations. The verifier rejects if the values for the two sides are different.

We summarize the above discussion via the following claim.

Claim 6.9. *In a $(\mathbb{F}, k, t)$-tensor IOP, where $\mathbf{V}$ has oracle access to $v \in \mathbb{F}^{k^t}$ and explicit input $v_1, \ldots, v_t \in \mathbb{F}^k$, $\mathbf{V}$ can check that $v = v_1 \otimes \cdots \otimes v_t$ with soundness error $t/|\mathbb{F}|$, using a single tensor query to v and $O(tk)$ arithmetic operations.*

6.3 Cyclic-shift test

We say that $b \in \mathbb{F}^{\ell \cdot k^t}$ is the *cyclic shift* of $a \in \mathbb{F}^{\ell \cdot k^t}$, which we denote $b = \mathrm{shift}(a)$, if $b_{\ell \cdot k^t} = a_1$ and, for all $i \in [\ell \cdot k^t - 1]$, it holds that $a_i = b_{i-1}$. In Construction 6.13 and Construction 6.17, $\mathbf{V}$ must check that $\mathbf{P}$ has sent $a, b \in \mathbb{F}^{\ell \cdot k^t}$ such that $b = \mathrm{shift}(a)$.

Consider the polynomials $A(X) := \sum_{i=1}^{\ell \cdot k^t} a_i X^{i-1}$ and $B(X) := \sum_{i=1}^{\ell \cdot k^t} b_i X^{i-1}$. One can verify that the polynomial identity $A(X) - X \cdot B(X) = (1 - X^{\ell \cdot k^t}) \cdot b_{\ell \cdot k^t}$ holds if and only if $b = \mathrm{shift}(a)$. Therefore, it suffices for the verifier to test this identity at a random evaluation point $\gamma \in \mathbb{F}$; indeed, by the Schwartz–Zippel lemma, if $b \neq \mathrm{shift}(a)$, then the probability that $A(\gamma) - \gamma \cdot B(\gamma) = (1 - \gamma^{\ell \cdot k^t}) \cdot b_{\ell \cdot k^t}$ is at most $\ell \cdot k^t/|\mathbb{F}|$.

We now explain how the verifier can evaluate the above expression at a given $\gamma \in \mathbb{F}$. We define:

$$\gamma_0 := (1, \gamma^{k^t}, \ldots, \gamma^{k^t(\ell-1)}) \quad \text{and, for every } i \in [t], \quad \gamma_i := (1, \gamma^{k^{t-i}}, \ldots, \gamma^{k^{t-i}(k-1)}) \ .$$

Observe that $\gamma_0 \otimes \gamma_1 \otimes \cdots \otimes \gamma_t = (1, \gamma, \ldots, \gamma^{\ell \cdot k^t - 1})$. Thus, $\mathbf{V}$ can use the tensor query $(\gamma_0, \gamma_1, \ldots, \gamma_t)$ to evaluate each of $A(X)$ and $B(X)$ at γ and another tensor query to obtain $b_{\ell \cdot k^t}$, and then check the equation $A(\gamma) - \gamma \cdot B(\gamma) = (1 - \gamma^{\ell \cdot k^t}) \cdot b_{\ell \cdot k^t}$.

We summarize the above discussion via the following claim.

Claim 6.10. *In a $(\mathbb{F}, k, t)$-tensor IOP, where $\mathbf{V}$ has oracle access to $a, b \in \mathbb{F}^{\ell \cdot k^t}$, $\mathbf{V}$ can check that $b = \mathrm{shift}(a)$ with soundness error $\ell \cdot k^t/|\mathbb{F}|$, using 3 tensor queries and $O(tk)$ arithmetic operations.*

6.4 Entry-product protocol

We say that $\tau \in \mathbb{F}$ is the *entry product* of a vector $a \in \mathbb{F}^{\ell \cdot k^t}$, denoted $\tau = \mathrm{prod}(a)$, if $\tau = \prod_{i \in [\ell \cdot k^t]} a_i$. As part of the look-up protocol in Construction 6.17, $\mathbf{V}$ must check that $\mathbf{P}$ has correctly claimed the entry-product of various vectors. Below we describe an entry-product protocol to enable this.

Definition 6.11. *The* **entry-product** *indexed relation R_{EP} is the set of tuples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, \ell, \tau), a)$ where $a \in \mathbb{F}^{\ell \cdot k^t}$ and $\tau = \mathrm{prod}(a)$.*

Lemma 6.12. *For every finite field $\mathbb{F}$ and positive integers k, t, Construction 6.13 is an exact $(\mathbb{F}, k, t)$-tensor IOPP for the indexed relation R_{EP} that has the following parameters: soundness error is $O(\ell \cdot k^t / |\mathbb{F}|)$; round complexity is $O(\log(\ell \cdot k^t))$; proof length is $O(\ell \cdot k^t)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(\ell \cdot k^t)$ field operations; and the verifier uses $O(\ell + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i} = \bot$ is the index itself.*

The construction uses three vectors $c, d, e \in \mathbb{F}^{\ell \cdot k^t}$ of partial products of the entries of a such that $a \circ c = d$, $e = \mathrm{shift}(c)$ with $e_{\ell \cdot k^t} = 1$, and d is almost equal to e except that $d_{\ell \cdot k^t} = \tau$. Together, these conditions imply that $\tau = \mathrm{prod}(a)$. The prover and verifier then use a Hadamard-product protocol and the cyclic-shift test described in Section 6.3 to check that all of these conditions are satisfied.

Construction 6.13 (entry-product)**.** We construct an exact interactive oracle proof of proximity $\mathsf{IOPP} = (\mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{EP}. There is no indexer $\mathbf{I}$ because R_{EP} always has $\mathbb{i} = \bot$. The prover $\mathbf{P}$ takes as input an index $\mathbb{i}$, instance $\mathbb{x} = (\mathbb{F}, \ell, \tau)$, and witness $\mathbb{w} = a$; the verifier $\mathbf{V}$ has (tensor) query access to the index $\mathbb{i}$ and witness $\mathbb{w}$ and takes as input the instance $\mathbb{x}$.

- The prover $\mathbf{P}$ computes the partial products $\{\prod_{i \le \kappa} a_\kappa\}_{i \in [\ell \cdot k^t]}$. Let $c, d, e \in \mathbb{F}^{\ell \cdot k^t}$ be defined as follows:

$$
\begin{aligned}
c &:= (1, a_1, a_1 \cdot a_2, \ldots, \prod_{i=1}^{\ell \cdot k^t - 1} a_\kappa) \ , \\
d &:= (a_1, a_1 \cdot a_2, \ldots, \prod_{i=1}^{\ell \cdot k^t} a_\kappa) \ , \\
e &:= (a_1, a_1 \cdot a_2, \ldots, \prod_{i=1}^{\ell \cdot k^t - 1} a_\kappa, 1) \ .
\end{aligned}
$$

  The prover computes $(c, d, e) \in \mathbb{F}^{3\ell \cdot k^t}$ and sends them to the verifier.

- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in a Hadamard-product protocol with $\mathbb{x} = (\mathbb{F}, \ell \cdot k^t)$ and $\mathbb{w} = (a, c, d)$ to show that $a \circ c = d$.

- The verifier $\mathbf{V}$ makes three queries to check that $e = \mathrm{shift}(c)$. In the process, the verifier learns $e_{\ell \cdot k^t}$, and rejects if it is not equal to 1. Moreover, the verifier $\mathbf{V}$ samples random $\eta = (s_0, s_1, \ldots, s_t) \in \mathbb{F}^\ell \times (\mathbb{F}^k)^t$ and then checks that $\langle \otimes \eta, d \rangle - \langle \otimes \eta, e \rangle = (s_0)_\ell \cdot (s_1)_k \cdot \ldots \cdot (s_t)_k \cdot (\tau - 1)$.

Properties of Construction 6.13. We omit a formal proof of the parameters claimed in Lemma 6.12, as they follow readily from the properties of the cyclic-shift test in Section 6.3 and the properties of the Hadamard-product protocol stated in Lemma 6.12.

6.5 Look-up protocol

We give a protocol for R_{LU} (see Definition 6.6) that supports instances of R_{LU} with $N = k^t$ and $M = \ell \cdot k^t$.

Lemma 6.14. *For every finite field $\mathbb{F}$ and positive integers k, t, Construction 6.17 is an exact $(\mathbb{F}, k, t)$-tensor holographic IOPP for the indexed relation R_{LU} that supports instances over $\mathbb{F}$ with $M = \ell \cdot k^t$ and $N = k^t$ and has the following parameters: soundness error is $O((M+N)/|\mathbb{F}|)$; round complexity is $O(\log((M+N)))$; proof length is $O(M+N)$ elements in $\mathbb{F}$; query complexity is $O(1)$; the prover uses $O(M+N)$ field operations; and the verifier uses $O(\ell + tk)$ field operations. Moreover, the indexer algorithm is degenerate: the encoding of an index $\mathbb{i}$ is the index itself.*

Our look-up protocol is based on a bivariate polynomial identity from [GW20]. We provide a definition and state the polynomial identity; after that we describe our look-up protocol. In Section 2.5 we gave a simplified description in the case that the entries of b were distinct. Here, we describe the general case by replacing the "sort" operation with the more general "merge" operation.

Definition 6.15. *Let $a \in \mathbb{F}^M$ and $b \in \mathbb{F}^N$ with $a \subseteq b$. We define $\mathrm{merge}_b(a)$ to be the set of all vectors $w \in \mathbb{F}^{N+M}$ obtained as follows: initialize a list w to be equal to b; then, for each $j \in [M]$, insert a_j into w next to* any *entry w_i of w such that $a_j = w_i$. (At least one such entry exists because $a \subseteq b$.)*

Lemma 6.16. *Let $a \in \mathbb{F}^M$ and $b \in \mathbb{F}^N$. Then $a \subseteq b$ if and only if there exists $w \in \mathbb{F}^{N+M}$ such that*

$$\prod_{j=1}^{M+N} \big(Y(1+Z) + w_j + \mathrm{shift}(w)_j \cdot Z\big) = (1+Z)^M \prod_{j=1}^{M} (Y + a_j) \prod_{j=1}^{N} \big(Y(1+Z) + b_j + \mathrm{shift}(b)_j \cdot Z\big) \ . \quad (15)$$

In the case that $a \subseteq b$, we may take any $w \in \mathrm{merge}_b(a)$ to satisfy Equation (15).

Proof. The lemma is a minor modification of [GW20, Claim 3.1] that incorporates "wrap-around" in the products involving the entries of w and b, and accounts for the fact that entries in b may not be distinct. □

The verifier has tensor-query access to the vectors $I \in \mathbb{F}^M$ and $[N] \in \mathbb{F}^N$. The first step in the protocol is to use algebraic hashing to compress each of the pairs (c, I) and $(d, [N])$ into a single vector. Taking a linear combination with a random challenge ζ, we get two new vectors $a := (c + \zeta \cdot I) \in \mathbb{F}^M$ and $b := (d + \zeta \cdot [N]) \in \mathbb{F}^N$. The condition $(c, I) \subseteq (d, [k^t])$ is now replaced with the condition $a \subseteq b$, which is equivalent up to a small soundness error over the choice of ζ.

The prover will also construct a suitable vector $w \in \mathrm{merge}_b(a)$. Importantly, one can do this in linear time using linear scans. Having sent w to the verifier, in the remainder of the protocol, the prover recomputes Equation (15) at random evaluation points chosen by the verifier and sends intermediate computation steps to the verifier. Both parties run shift and entry-product subprotocols to check that the computation was performed correctly and the evaluated polynomial identity holds.

Note that while taking $\mathbb{F}$-linear combinations of vectors of indices $[N]$ and I above, we are writing integers from $[N]$ as field elements. This does not mean that we require $N < \mathrm{char}(\mathbb{F})$. Rather, we implicitly consider some injective mapping of $[N]$ into $\mathbb{F}$. (In particular, this works even if $\mathrm{char}(\mathbb{F}) = 2$, for example.)

Construction 6.17 (look-up)**.** We construct an exact holographic interactive oracle proof $\mathsf{IOPP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{LU}. Given as input an index $\mathbb{i} = (I, N)$, the indexer $\mathbf{I}$ outputs the oracle message $\Pi_0 := (I, [N])$. The prover $\mathbf{P}$ takes as input the index $\mathbb{i}$, instance $\mathbb{x} = (\mathbb{F}, M, N)$, and witness $\mathbb{w} = (c, d)$; the verifier $\mathbf{V}$ has query access to the index $\mathbb{i}$ and the witness $\mathbb{w}$ and takes as input the instance $\mathbb{x}$.

- The verifier $\mathbf{V}$ sends a random challenge $\zeta \in \mathbb{F}$ to $\mathbf{P}$.
- The prover $\mathbf{P}$ computes $a := c + \zeta \cdot I \in \mathbb{F}^M$ and $b := d + \zeta \cdot [N] \in \mathbb{F}^N$. Next, $\mathbf{P}$ constructs the vector $w \in \mathbb{F}^{N+M}$ that contains the same entries as (a, b) with the each entry of the form $b_i = d_i + \zeta \cdot i$ ordered by the index $i \in [N]$ that contributes to that entry. In more detail, $\mathbf{P}$ does the following:
  - scan $(I, [N]) \in \mathbb{F}^{N+M}$ to deduce the frequency vector $f \in \mathbb{N}^N$ where f_i is the number of occurrences of the index $i \in [N]$ in $(I, [N])$;
  - allocate the zero vector $w := 0^{N+M}$ and then perform a linear scan on w, where for each $i \in [N]$, inserting f_i copies of $d_i + \zeta \cdot i$ into w.

  Then $\mathbf{P}$ sets $b_\circlearrowleft := \mathrm{shift}(b)$ and $w_\circlearrowleft := \mathrm{shift}(w)$, and sends the oracle message $(b_\circlearrowleft, w, w_\circlearrowleft) \in \mathbb{F}^{(2\ell+3)\cdot k^t}$.
- The verifier $\mathbf{V}$ sends random challenges $\gamma, \beta \in \mathbb{F}$ to $\mathbf{P}$.
- The prover $\mathbf{P}$ computes the following vectors:

$$a^* := \gamma \cdot 1^M + a \ , \tag{16}$$

$$b^* := \gamma(1+\beta) \cdot 1^N + b + \beta \cdot b_\circlearrowleft \ , \tag{17}$$

$$w^* := \gamma(1+\beta) \cdot 1^{N+M} + w + \beta \cdot w_\circlearrowleft \ . \tag{18}$$

  Next, $\mathbf{P}$ computes the entry products $\chi_{a^*} := \mathrm{prod}(a^*)$, $\chi_{b^*} := \mathrm{prod}(b^*)$, and $\chi_{w^*} := \mathrm{prod}(w^*)$. Then $\mathbf{P}$ sends the oracle message $(a^*, b^*, w^*) \in \mathbb{F}^{2(N+M)}$ and non-oracle message $(\chi_{a^*}, \chi_{b^*}, \chi_{w^*}) \in \mathbb{F}^3$ to $\mathbf{V}$.
- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in several sub-protocols.
  - An entry-product protocol with $\mathbb{x} = (\mathbb{F}, M, \chi_{a^*})$ and $\mathbb{w} = a^*$ to show that $\chi_{a^*} = \mathrm{prod}(a^*)$.
  - An entry-product protocol with $\mathbb{x} = (\mathbb{F}, N, \chi_{b^*})$ and $\mathbb{w} = b^*$ to show that $\chi_{b^*} = \mathrm{prod}(b^*)$.
  - An entry-product protocol with $\mathbb{x} = (\mathbb{F}, N+M, \chi_{w^*})$ and $\mathbb{w} = w^*$ to show that $\chi_{w^*} = \mathrm{prod}(w^*)$.
- The verifier $\mathbf{V}$ checks that $\chi_{w^*} = (1+\beta)^M \cdot \chi_{a^*} \cdot \chi_{b^*}$. Next, $\mathbf{V}$ samples random tensor queries as follows:

$$\eta_1 = (s_0^{(1)}, s_1^{(1)}, \ldots, s_t^{(1)}) \in \mathbb{F}^\ell \times (\mathbb{F}^k)^t \ ,$$
$$\eta_2 = (s_1^{(2)}, \ldots, s_t^{(2)}) \in (\mathbb{F}^k)^t \ ,$$
$$\eta_3 = (s_0^{(3)}, s_1^{(3)}, \ldots, s_t^{(3)}) \in \mathbb{F}^{\ell+1} \times (\mathbb{F}^k)^t \ .$$

  Next, $\mathbf{V}$ performs the checks below, by using tensor queries to obtain each scalar-product term on the left-hand side of each equation and by directly computing the right-hand side using factorizations such as $\langle \otimes\eta_1, 1^N \rangle = \prod_{i=1}^t \langle s_1^{(i)}, 1^k \rangle$.

$$\langle \otimes\eta_1, a^* \rangle - \langle \otimes\eta_1, c \rangle - \zeta \cdot \langle \otimes\eta_1, I \rangle = \gamma \cdot \langle \otimes\eta_1, 1^M \rangle \qquad \text{which checks Eq. (16),}$$
$$\langle \otimes\eta_2, b^* \rangle - \langle \otimes\eta_2, d \rangle - \zeta \cdot \langle \otimes\eta_2, [k^t] \rangle - \beta \cdot \langle \otimes\eta_2, b_\circlearrowleft \rangle = y(1+\beta) \cdot \langle \otimes\eta_1, 1^N \rangle \qquad \text{which checks Eq. (17),}$$
$$\langle \otimes\eta_3, w^* \rangle - \langle \otimes\eta_3, w \rangle - \beta \cdot \langle \otimes\eta_3, w_\circlearrowleft \rangle = \gamma(1+\beta) \cdot \langle \otimes\eta_3, 1^{N+M} \rangle \qquad \text{which checks Eq. (18).}$$

  Finally, $\mathbf{V}$ checks that $b_\circlearrowleft = \mathrm{shift}(b)$ and $w_\circlearrowleft = \mathrm{shift}(w)$, using three queries for each check as described in Section 6.3.

Properties of Construction 6.17. Below in Lemma 6.18 we prove that Construction 6.17 has the soundness error stated in Lemma 6.14. Before that, we informally discuss (but omit formal proofs of) the other efficiency parameters claimed in Lemma 6.14. Construction 6.17 is an interactive reduction followed by various subprotocols, including scalar-product protocols invoked on vectors of length $M + N$. The interactive reduction itself introduces a soundness error of $O((M + N)/|\mathbb{F}|)$ into Construction 6.17, which dominates the contributions of all sub-protocols. The contributions of the scalar-product protocols dominate the values of all of the other efficiency parameters of Lemma 6.14. Thus, all of the other values in Lemma 6.14 apart from the soundness error come from substituting $n = M + N$ into Theorem 5.13.

Lemma 6.18. *Construction 6.17 has soundness error* $\epsilon_{\text{LU}} = (3M + 4N)/|\mathbb{F}|$.

Proof. Suppose that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((I, N), (\mathbb{F}, M, N), (c, d)) \notin L(R_{\text{LU}})$. Then $\{(c_i, I_i)\}_{i\in[M]} \not\subseteq \{(d_j, j)\}_{j\in[N]}$. In other words, there exists some $i \in [M]$ such that, for all $j \in [N]$, we have $(c_i, I_i) \neq (d_j, j)$.

Fix a malicious prover, which determines certain next-message functions for oracle messages $b_{\circlearrowleft}(\zeta)$, $w(\zeta)$, $w_{\circlearrowleft}(\zeta)$, $a^*(\zeta, \gamma, \beta)$, $b^*(\zeta, \gamma, \beta)$, $w^*(\zeta, \gamma, \beta)$; and non-oracle messages $\chi_{a^*}(\zeta, \gamma, \beta)$, $\chi_{b^*}(\zeta, \gamma, \beta)$, $\chi_{w^*}(\zeta, \gamma, \beta)$. Further, define vectors $a(\zeta) := c + \zeta \cdot I$ and $b(\zeta) := d + \zeta \cdot [N]$.

For each $j \in [N]$, there is at most one $\zeta \in \mathbb{F}$ such that $c_i + \zeta \cdot I_i = d_j + \zeta \cdot j$. Thus, for a uniformly random $\zeta \in \mathbb{F}$, the probability that $c_i + \zeta \cdot I_i = d_j + \zeta \cdot j$ is at most $1/|\mathbb{F}|$. By a union bound, the probability over the random choice of ζ that there exists some $j \in [N]$ such that $c_i + \zeta \cdot I_i$ is equal to $d_j + \zeta \cdot j$ is at most $N/|\mathbb{F}|$. Hence, except with probability $N/|\mathbb{F}|$ over the random choice of ζ, we have $a \not\subseteq b$ and so by Lemma 6.16 the polynomial identity of Equation (15) does not hold.

Evaluating the polynomial identity at random $\beta, \gamma \in \mathbb{F}$, observe that by the Schwartz–Zippel lemma, except with probability at most $2(M + N)/|\mathbb{F}|$, we have

$$\prod_{j=1}^{M+N} (\gamma(1 + \beta) + w_j + \beta \cdot \text{shift}(w)_j) \neq (1 + \beta)^M \prod_{j=1}^{M} (\gamma + a_j) \prod_{j=1}^{N} (\gamma(1 + \beta) + b_j + \beta \cdot \text{shift}(b)_j) \ .$$

This implies that one of the following conditions holds.

- $b_{\circlearrowleft} \neq \text{shift}(b)$. By Claim 6.10, the verifier's cyclic-shift test rejects, except with probability at most $N/|\mathbb{F}|$.
- $w_{\circlearrowleft} \neq \text{shift}(w)$. Similar to the previous case, with rejection except for a soundness error of $(N + M)/|\mathbb{F}|$.
- $a^* \neq \gamma \cdot 1^M + c + \zeta \cdot I$. With $\eta_1 = (s_0^{(1)}, s_1^{(1)}, \ldots, s_t^{(1)})$, we apply the Schwartz–Zippel Lemma to the non-zero polynomial $P(\eta_1) := \langle \otimes\eta_1, a^* \rangle - \langle \otimes\eta_1, c \rangle - \zeta \cdot \langle \otimes\eta_1, I \rangle - \gamma \cdot \langle \otimes\eta_1, 1^M \rangle$ of total degree $t + 1$, concluding that $\Pr_{\eta_1}[P(\eta_1) = 0] \leq (t + 1)/|\mathbb{F}|$. Whenever $P(\eta_1) \neq 0$, the verifier rejects.
- $b^* \neq \gamma(1 + \beta) \cdot 1^N + d + \zeta \cdot [N] + \beta \cdot b_{\circlearrowleft}$ *or* $w^* \neq \gamma(1 + \beta) \cdot 1^{N+M} + w + \beta \cdot w_{\circlearrowleft}$. Similar to the previous case, with rejection except for a soundness error of $t/|\mathbb{F}|$ or $(t + 1)/|\mathbb{F}|$ respectively.
- $\chi_{a^*} \neq \text{prod}(a^*)$. By the soundness of the entry-product protocol, the verifier accepts with probability at most $\epsilon_{\text{EP}}(M)$.
- $\chi_{b^*} \neq \text{prod}(b^*)$ *or* $\chi_{w^*} \neq \text{prod}(w^*)$. Similar to the previous case, leading to soundness error $\epsilon_{\text{EP}}(N)$ or $\epsilon_{\text{EP}}(N + M)$.
- $\chi_{w^*} \neq (1 + \beta)^M \cdot \chi_{a^*} \cdot \chi_{b^*}$. The verifier checks this directly, and will therefore reject.

A union bound shows that $\epsilon_{\text{LU}} = \frac{N}{|\mathbb{F}|} + \frac{2(M+N)}{|\mathbb{F}|} + \max\{\frac{M+N}{|\mathbb{F}|}, \epsilon_{\text{EP}}(N), \epsilon_{\text{EP}}(M), \epsilon_{\text{EP}}(N + M)\}$. □

7 Preliminaries on codes

We introduce preliminaries on error-correcting codes that will be used in our transformation from a tensor IOP to a standard IOP (see Section 8 for the result, and Sections 9 and 10 for the construction and its analysis).

We consider functions $c\colon [n] \to \mathbb{F}$ for a given domain $[n]$ and finite field $\mathbb{F}$. The Hamming distance $d(c, c')$ between two functions $c, c'\colon [n] \to \mathbb{F}$ is the number of inputs for which c and c' differ; the relative distance is $\delta(c, c') := d(c, c')/n$. The Hamming weight of c is the number of inputs for which c is non-zero.

A linear error-correcting code $\mathcal{C}$ over $\mathbb{F}$ is a set of functions $c\colon [n] \to \mathbb{F}$ that form an $\mathbb{F}$-linear space. The block length is n and the message length is the rank k of the linear space $\mathcal{C}$; the rate is $\rho := k/n$. The minimum distance $d = d(\mathcal{C})$ is the minimal Hamming distance between any two distinct codewords c and c' in $\mathcal{C}$. The relative minimum distance δ of the code is defined similarly.

When $d(c, \mathcal{C})$ is smaller than the unique decoding radius of $\mathcal{C}$ (which is $d(\mathcal{C})/2$) there is a unique closest codeword $\overline{c} \in \mathcal{C}$; otherwise we default to setting $\overline{c} := 0$.

Codewords can also be interpreted as vectors in $\mathbb{F}^n$. Messages from $\mathbb{F}^k$ can be encoded using an injective linear function $\mathrm{Enc}\colon \mathbb{F}^k \to \mathcal{C}$: a message $f \in \mathbb{F}^k$ is mapped to fG, where $G \in \mathbb{F}^{k \times n}$ is the code's generator matrix (the matrix whose rows generate $\mathcal{C}$ as a linear subspace of $\mathbb{F}^n$). In general, the cost of encoding f by computing fG is $O(kn)$ operations (or, more generally, is linear in the number of non-zero entries in G).

In this work, we use error-correcting codes where messages can be encoded in linear time, i.e., for which Enc is computable via an $\mathbb{F}$-arithmetic circuit with $O(k)$ gates. The following result of [DI14] gives a construction of such codes, generalizing the seminal work of [Spi96] to any finite field and drawing on techniques from [GI01] to achieve the Gilbert–Varshamov bound.

Lemma 7.1 ([DI14])**.** *Let $\mathbb{F}$ be a finite field with q elements. For every rate parameter $\rho \in (0, 1)$ there is a polynomial-time constructible circuit family $E_k\colon \mathbb{F}^k \times \mathbb{F}^{\nu(k)} \to \mathbb{F}^n$ such that:*

- *E_k is an $\mathbb{F}$-arithmetic circuit of size $O(k)$ with $\nu(k) = O(k)$ and $k = \lfloor \rho n \rfloor$;*
- *for every $r \in \mathbb{F}^{\nu(k)}$, $E_k(\cdot, r)$ is an injective linear encoding function with message length k and block length n;*
- *for every $\epsilon > 0$, and letting H_q denote the q-ary entropy function,*

$$\Pr_{r \leftarrow \mathbb{F}^{\nu(k)}} \Big[\textit{the relative distance of } E_k(\cdot, r) \textit{ is at least } H_q^{-1}(1-\rho) - \epsilon \Big] \geq 1 - q^{-\Omega(\epsilon n)} \ .$$

Interleaved codes. We say that c is an *interleaved codeword* if, for some $\ell \in \mathbb{N}$, we have $c \in \mathcal{C}^\ell$. We equivalently view c as a function from $[\ell] \times [n]$ to $\mathbb{F}$ or as a matrix in $\mathbb{F}^{\ell \times n}$.

We consider a single symbol of a function $c\colon [\ell] \times [n] \to \mathbb{F}$ to be a column of the $\ell \times n$ matrix that represents c (the column thus contains ℓ elements of $\mathbb{F}$). Accordingly, we define the block-wise relative distance between two functions $c, c'\colon [\ell] \times [n] \to \mathbb{F}$ to be the number of columns in which c and c' differ. If c is within the unique decoding radius of $\mathcal{C}^\ell$, we denote by $\overline{c}$ the unique codeword in $\mathcal{C}^\ell$ that is closest to c (and otherwise we default to setting $\overline{c} := 0$); note that $\overline{c}$ is obtained by replacing each row of c with the corresponding closest codeword in $\mathcal{C}$.

Moreover, we define the block-wise relative distance of a set of functions $\{c^{(s)}\colon [\ell] \times [n] \to \mathbb{F}\}_s$ to $\mathcal{C}$ to be the fraction of columns where at least one of the functions deviates from a codeword as follows:

$$\Delta(\{c^{(s)}\}_s, \mathcal{C}) := \frac{\left|\left\{ j \in [n] \;\middle|\; \exists\, (s, i) \text{ s.t. } c^{(s)}(i, j) \neq \overline{c}^{(s)}(i, j) \right\}\right|}{n} \ .$$

Definition 7.2. *The* **stacking** *of functions* $\{c^{(s)}\colon [a_s] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}\}_s$ *is the function*

$$\mathrm{Stack}\left(\{c^{(s)}\}_s\right) : [\textstyle\sum_s a_s] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}$$

such that $\mathrm{Stack}\left(\{c^{(s)}\}_s\right)(i, j_1, \ldots, j_h) = c^{(\ell)}(i - \sum_{u \le \ell} a_u, j_1, \ldots, j_h)$ *for* $i \in \{1 + \sum_{u \le \ell} a_u, \ldots, \sum_{u \le \ell+1} a_u\}$.

We introduce notation for taking linear combinations of interleaved words and functions, an operation that we refer to as "folding" and use throughout the next few sections.

Definition 7.3. *The* **folding** *of a function* $c\colon [a] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ *by a linear combination* $\alpha\colon [a] \to \mathbb{F}$ *is the function* $\mathrm{Fold}(c; \alpha)\colon [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ *defined as follows:*

$$\mathrm{Fold}(c; \alpha) := \sum_{i \in [a]} \alpha(i) c(i, \cdot, \ldots, \cdot) \ . \tag{19}$$

Moreover, the folding of a set of functions $\{c^{(s)}\}_s$ *by a set of corresponding linear combinations* $\{\alpha^{(s)}\}_s$ *is the function* $\mathrm{Fold}(\{c^{(s)}\}_s; \{\alpha^{(s)}\}_s)\colon [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ *defined as follows:*

$$\mathrm{Fold}(\{c^{(s)}\}_s; \{\alpha^{(s)}\}_s) := \sum_s \mathrm{Fold}(c^{(s)}; \alpha^{(s)}) \ .$$

More generally, for $r \in \{0, 1, \ldots, h\}$*, we write* Fold_r *to indicate a folding operation that is applied to the* r*-th coordinate in the sum in Equation* (19)*, as opposed to the* 0*-th coordinate.*

Tensor-product codes. The tensor product code $\mathcal{C}^{\otimes t}$ is the linear code in $\mathbb{F}^{n^t}$ with message length k^t, block length n^t, and distance d^t that comprises all functions $c\colon [n]^t \to \mathbb{F}$ whose restriction to any axis-parallel line is in $\mathcal{C}$. Namely, for every $j \in [t]$ and $a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_t \in [n]$, the function $c'\colon [n] \to \mathbb{F}$ defined by $c'(i) := c(a_1, \ldots, a_{j-1}, i, a_{j+1}, \ldots, a_t)$ is in $\mathcal{C}$. The encoding function associated to $\mathcal{C}^{\otimes t}$ is defined below.

Definition 7.4. *Let* $G\colon [k] \times [n] \to \mathbb{F}$ *be the generator matrix of a linear code* $\mathcal{C}$ *in* $\mathbb{F}^n$*, and* $f\colon [k]^t \to \mathbb{F}$ *a message function with inputs indexed by* $(i_1, \ldots, i_t)$*. The* $\mathcal{C}^{\otimes t}$**-encoding** *of* f *is the function* $\mathrm{Enc}_{1,\ldots,t}(f)\colon [n]^t \to \mathbb{F}$*, with inputs indexed by* $(j_1, \ldots, j_t)$*, defined as follows:*

$$\mathrm{Enc}_{1,\ldots,t}(f)(j_1, \ldots, j_t) := \sum_{i_1, \ldots, i_t \in [k]} f(i_1, \ldots, i_t) G(i_1, j_1) \cdots G(i_t, j_t) \ .$$

Lemma 7.5. *If the encoding function* $\mathrm{Enc}\colon \mathbb{F}^k \to \mathcal{C}$ *of a code* $\mathcal{C}$ *has arithmetic complexity* $\theta(k) \cdot k$*, then the encoding function* $\mathrm{Enc}_{1,\ldots,t}\colon \mathbb{F}^k \to \mathcal{C}^{\otimes t}$ *of the tensor code* $\mathcal{C}^{\otimes t}$ *has arithmetic complexity* $\frac{\rho^{-t}-1}{\rho^{-1}-1}\theta(k) \cdot k^t$*. (In particular, if* $\mathcal{C}$ *is linear-time encodable then so is the tensor code* $\mathcal{C}^{\otimes t}$*.)*

Proof sketch. View a message in $\mathbb{F}^{k^t}$ as k^{t-1} vectors in $\mathbb{F}^k$. Encode each of the vectors to get a partial encoding with $k^{t-1}n$ elements. View it as $k^{t-2}n$ vectors in $\mathbb{F}^k$ and encode each of them to obtain a partial encoding with $k^{t-2}n^2$ elements. Compute t partial encodings to obtain the full encoding.

After performing r^* partial encodings, there are $k^{t-r^*}n^{r^*} = k^t\rho^{-r^*}$ elements of $\mathbb{F}$, which are then viewed as $k^{t-1}\rho^{-r^*}$ vectors of $\mathbb{F}^k$ before computing the next partial encoding. In this way, the final encoding can be obtained via $\theta k^t + \theta k^t\rho^{-1} + \cdots + \theta k^t\rho^{-t+1} = (1 + \rho^{-1} + \cdots + \rho^{-t+1})\theta k^t$ operations. Summing the geometric series yields the claimed arithmetic complexity. □

In later sections we will also need more general notions of encodings, which consider *partial encodings* of a high-dimensional array along different axis-parallel lines. We provide these definitions below.

Definition 7.6. *Let $G\colon [k] \times [n] \to \mathbb{F}$ be the generator matrix of a linear code $\mathcal{C}$ in $\mathbb{F}^n$. The* **encoding** *of a function $c\colon [k] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ is the function $\mathrm{Enc}(c)\colon [n] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ defined as follows:*

$$\mathrm{Enc}(c)(j, \cdot, \ldots, \cdot) := \sum_{i \in [k]} c(i, \cdot, \ldots, \cdot) G(i, j) \ .$$

More generally, we write Enc_r to indicate an encoding operation that is applied to the r-th coordinate.

Definition 7.7. *Given a subset $S \subseteq [t]$, the S-***encoding** *of a function $f\colon [k]^t \to \mathbb{F}$ is the mapping defined by applying Enc_r (i.e., encoding f along the r-th coordinate) for every $r \in S$. This mapping is well defined as the encodings $\{\mathrm{Enc}_r\}_{r \in [t]}$ commute with one another as they each operate on different coordinates.*

We will use the convention that an index "i" represents an unencoded axis while an index "j" represents an encoded axis: if a function indexed by i_r is encoded using Enc_r, then the new function is indexed by j_r.

Commutativity. We will use the fact that folding commutes with partial encodings, as stated below.

Lemma 7.8. *Consider functions*

$$\begin{aligned}
f_{r-1}&\colon [a] \times [k]^{t-r} \to \mathbb{F} && \text{indexed by } (i_{r-1}, \ldots, i_t)\\
c_{r-1}&\colon [a] \times [n]^{t-r} \to \mathbb{F} && \text{indexed by } (i_{r-1}, j_r, \ldots, j_t)\\
f_r&\colon [a] \times [k]^{t-r-1} \to \mathbb{F} && \text{indexed by } (i_r, \ldots, i_t)\\
c_r&\colon [a] \times [n]^{t-r-1} \to \mathbb{F} && \text{indexed by } (i_r, j_{r+1}, \ldots, j_t)\\
q&\colon [a] \to \mathbb{F} && \text{indexed by } i_{r-1}
\end{aligned}$$

such that the following conditions hold:

$$c_{r-1} = \mathrm{Enc}_{r,\ldots,t}(f_{r-1}) \ , \quad f_r = \mathrm{Fold}_{r-1}(f_{r-1}; q) \ , \quad c_r = \mathrm{Enc}_{r+1,\ldots,t}(f_r) \ . \tag{20}$$

Then $\mathrm{Fold}_{r-1}(c_{r-1}; q) = \mathrm{Enc}_r(c_r)$.

Proof. The definitions of multiple encodings (Definition 7.7) and folding (Definition 7.3) directly yield the following derivation:

$$\begin{aligned}
\mathrm{Fold}_{r-1}(c_{r-1}; q)(j_r, \ldots, j_t) &= \sum_{i_{r-1} \in [a]} q(i_{r-1}) c_{r-1}(i_{r-1}, j_r, \ldots, j_t)\\
&= \sum_{i_{r-1} \in [a]} q(i_{r-1}) \mathrm{Enc}_{r,\ldots,t}(f_{r-1})(i_{r-1}, j_r, \ldots, j_t)\\
&= \sum_{i_{r-1} \in [a]} q(i_{r-1}) \sum_{i_r, \ldots, i_t \in [k]} f_{r-1}(i_{r-1}, i_r, \ldots, i_t) G(i_r, j_r) \cdots G(i_t, j_t)\\
&= \sum_{i_r, \ldots, i_t \in [k]} G(i_r, j_r) \cdots G(i_t, j_t) \sum_{i_{r-1} \in [a]} q(i_{r-1}) f_{r-1}(i_{r-1}, i_r, \ldots, i_t)\\
&= \sum_{i_r, \ldots, i_t \in [k]} G(i_r, j_r) \cdots G(i_t, j_t) \mathrm{Fold}_{r-1}(f_{r-1}; q)(i_r, \ldots, i_t)
\end{aligned}$$

$$= \sum_{i_r,\ldots,i_t \in [k]} G(i_r, j_r) \cdots G(i_t, j_t) f_r(i_r, \ldots, i_t)$$
$$= \sum_{i_r \in [k]} G(i_r, j_r) \mathrm{Enc}_{r+1,\ldots,t}\left(f_r\right)(i_r, j_{r+1}, \ldots, j_t)$$
$$= \sum_{i_r \in [k]} G(i_r, j_r) c_r(i_r, j_{r+1}, \ldots, j_t) = \mathrm{Enc}_r(c_r)(j_r, \ldots, j_t) \ .$$

□

8 From point queries to tensor queries

Below is the formal restatement of Theorem 3, our transformation from a tensor IOP to a standard IOP.

Theorem 8.1. *There exists an explicit polynomial-time transformation* T *that satisfies the following. The inputs to the transformation are as follows:*

- *A linear code* $\mathcal{C}$ *over* $\mathbb{F}$ *with rate* $\rho = \frac{k}{n}$*, relative distance* $\delta = \frac{d}{n}$*, and encoding arithmetic complexity* $\theta(k) \cdot k$*.*
- *A holographic interactive oracle proof* $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *with queries in* $\mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ *for an indexed relation* R *with: soundness error* ϵ*; round complexity* rc*; proof length* $\mathsf{l} = \mathsf{li} + \mathsf{lp} = \ell k^t$*; query complexity* q*; arithmetic complexity* ti *for the indexer; arithmetic complexity* tp *for the prover; arithmetic complexity* tv *for the verifier.*

The output of the transformation $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}}) := \mathsf{T}(\mathcal{C}, \mathsf{IOP})$ *is an interactive oracle proof with queries in* $\mathcal{Q}_{\mathrm{point}}$ *for the indexed relation* R *with the following parameters:*

- *soundness error* $\epsilon + O\left(\frac{d^t}{|\mathbb{F}|}\right) + O\left((1-\delta^t/2)^\lambda\right)$*;*
- *round complexity* $\mathsf{rc} + t + 1$*;*
- *proof length* $O_{\rho,t}(\mathsf{q} \cdot \mathsf{l})$*;*
- *query complexity* $O(\lambda \cdot (\ell + kt \cdot \mathsf{q}))$*;*
- *indexer time* $\mathsf{ti} + O_{\rho,t}(\mathsf{li}) \cdot \theta(k)$*;*
- *prover time* $\mathsf{tp} + O_{\rho,t}(\mathsf{q} \cdot \mathsf{l}) \cdot \theta(k)$*; and*
- *verifier time* $\mathsf{tv} + O(\lambda \cdot (\ell + \theta(k) \cdot kt) \cdot \mathsf{q})$.

Remark 8.2. The transformation T receives input $\mathcal{C}$ in the form of a circuit for its encoding function, $\mathbf{I}$ represented as an algorithm, and $\mathbf{P}$ and $\mathbf{V}$ as a list of algorithms for their next-message functions.

Remark 8.3. The round complexity is $\mathsf{rc} + t + 1$ in the case that the verifier $\mathbf{V}$ makes non-adaptive queries, because then $\hat{\mathbf{V}}$ can handle all of the queries in a single round. In the case where the q queries are made as adaptively as possible, the round complexity would be $\mathsf{rc} + t + \mathsf{q}$. If $\mathbf{V}$ is public-coin, then $\hat{\mathbf{P}}$ already knows all of the queries and the round complexity would be $\mathsf{rc} + t$.

Theorem 8.1 is a consequence of two lemmas given in the next two sections: Lemma 9.1 in Section 9, which simulates a tensor IOP via standard IOP, given any IOP of proximity for tensor codes; and Lemma 10.1 in Section 10, which is a new IOP of proximity for tensor codes that we plug into the previous lemma.

The aforementioned proximity test is for the relation $R_\otimes$ defined below. Note that the relation has a "multi-part" witness with functions over different domains that must belong to tensor codes of different sizes. Below we also define a certain distance metric $\Delta_\otimes$ relative to which our proximity test will work.

Definition 8.4. *The indexed relation* $R_\otimes$ *is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \left(\bot, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t), (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)\right)$$

such that $c_0^{(0)} \in (\mathcal{C}^{\otimes t})^\ell$ *and, for all* $r \in [t-1]$ *and* $s \in [\mathsf{q}]$*, we have* $c_r^{(s)} \in (\mathcal{C}^{\otimes t-r})^k$*.*

Definition 8.5. *Let* $\mathbb{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ *be such that* $c_0^{(0)} \in \mathbb{F}^{\ell \cdot n^t}$ *and, for all* $r \in [t-1]$ *and* $s \in [\mathsf{q}]$*, we have* $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$*. Given* $(\mathbb{i}, \mathbb{x}) = (\bot, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t))$*, the* $\Delta_\otimes$*-distance of* $\mathbb{w}$ *to* $R_\otimes|_{(\mathbb{i},\mathbb{x})}$ *is*

$$\Delta_\otimes\left(\mathbb{w}, R_\otimes|_{(\mathbb{i},\mathbb{x})}\right) := \max\{\Delta_0, \Delta_1, \ldots, \Delta_{t-1}\} \quad \textit{where} \quad \begin{cases} \Delta_0 := \Delta(c_0^{(0)}, \mathcal{C}^{\otimes t}) \\ \forall\, r \in [t-1]\,,\ \Delta_r := \Delta(\{c_r^{(s)}\}_s, \mathcal{C}^{\otimes t-r}) \end{cases}.$$

9 Consistency checks on tensor codes

Lemma 9.1. *There exists an explicit polynomial-time transformation* T *that satisfies the following. The inputs to the transformation are as follows:*

- *A linear code* $\mathcal{C}$ *over* $\mathbb{F}$ *with rate* $\rho = \frac{k}{n}$*, relative distance* $\delta = \frac{d}{n}$*, and encoding arithmetic complexity* $\theta(k) \cdot k$*.*
- *An interactive oracle proof of proximity* $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$ *with queries in* $\mathcal{Q}_{\text{point}}$ *for the indexed relation* $R_{\otimes}(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$ *with soundness error* ϵ'*; round complexity* rc'*; proof length* l'*; query complexity* q'*; arithmetic complexity* tp' *for the prover; and arithmetic complexity* tv' *for the verifier.*
- *A holographic interactive oracle proof* $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *with queries in* $\mathcal{Q}_{\text{tensor}}(\mathbb{F}, k, t)$ *for an indexed relation* R *with: soundness error* ϵ*; round complexity* rc*; proof length* $\mathsf{l} = \mathsf{li} + \mathsf{lp} = \ell k^t$*; query complexity* q*; arithmetic complexity* ti *for the indexer; arithmetic complexity* tp *for the prover; and arithmetic complexity* tv *for the verifier.*

The output of the transformation $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}}) := \mathsf{T}(\mathcal{C}, \mathsf{IOP}, \mathsf{IOPP})$ *is an interactive oracle proof with queries in* $\mathcal{Q}_{\text{point}}$ *for the indexed relation* R *with the following parameters:*

- *soundness error* $\epsilon + \epsilon'(1/4) + \left(1 - \frac{\delta^t}{2}\right)^{\lambda}$*;*
- *round complexity* $\mathsf{rc} + \mathsf{rc}' + 1$*;*
- *proof length* $\mathsf{l}' + O_{\rho,t}(\mathsf{q} \cdot \mathsf{l})$*;*
- *query complexity* $\mathsf{q}' + O(\lambda \cdot (\ell + kt \cdot \mathsf{q}))$*;*
- *indexer time* $\mathsf{ti} + O_{\rho,t}(\mathsf{li}) \cdot \theta(k)$*;*
- *prover time* $\mathsf{tp} + \mathsf{tp}' + O_{\rho,t}(\mathsf{q} \cdot \mathsf{l}) \cdot \theta(k)$*; and*
- *verifier time* $\mathsf{tv} + \mathsf{tv}' + O(\lambda \cdot (\ell + \theta(k) \cdot kt) \cdot \mathsf{q})$.

Remark 9.2. The round complexity is $\mathsf{rc} + \mathsf{rc}' + 1$ in the case that the verifier $\mathbf{V}$ makes non-adaptive queries, because then $\hat{\mathbf{V}}$ can handle all of the queries in a single round. In the case where the q queries are made as adaptively as possible, the round complexity would be $\mathsf{rc} + \mathsf{rc}' + \mathsf{q}$. If $\mathbf{V}$ is public-coin, then $\hat{\mathbf{P}}$ already knows all of the queries and the round complexity would be $\mathsf{rc} + \mathsf{rc}'$.

Remark 9.3. The stated time for the (honest) prover $\hat{\mathbf{P}}$ assumes that $\hat{\mathbf{P}}$ has access to the output of the indexer $\mathbf{I}$, since $\hat{\mathbf{P}}$ must compute and send the answers to tensor queries involving the output of $\mathbf{I}$.

9.1 Construction

Though Definition 3.6 specifies that tensor queries are of the form $q(\mathbb{x}, \Pi_0, \ldots, \Pi_i) = \langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_j \rangle$, our transformation below handles tensor queries of the form $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \text{Stack}(\Pi_0, \ldots, \Pi_{\mathsf{rc}}) \rangle$, which are more general.

Construction 9.4. We describe the construction of $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$. The indexer $\hat{\mathbf{I}}$, given an index $\mathbb{i}$, runs $\mathbf{I}$ on $\mathbb{i}$ to produce $\Pi_0 \in \mathbb{F}^{\ell_0 \cdot k^t}$ indexed by $(i_0, i_1, \ldots, i_t) \in [\ell_0] \times [k]^t$; then computes and outputs $\hat{\Pi}_0 := \text{Enc}_{1,\ldots,t}(\Pi_0) \in \mathbb{F}^{\ell_0 \cdot n^t}$. The prover $\hat{\mathbf{P}}$ receives as input an instance $\mathbb{x}$ and witness $\mathbb{w}$, while the verifier $\hat{\mathbf{V}}$ receives as input the instance $\mathbb{x}$. The construction has two phases, a simulation phase and a consistency phase. We describe each in turn.

Simulation phase. For each $i \in [\mathsf{rc}]$, $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$ simulate the i-th round of the interaction between $\mathbf{P}(\mathbb{x}, \mathbb{w})$ and $\mathbf{V}(\mathbb{x})$, as well as any tensor queries by $\mathbf{V}$ to the received proof strings.

1. *Prover messages.* $\hat{\mathbf{P}}$ receives from $\mathbf{P}$ a proof message $\Pi_i \in \mathbb{F}^{\ell_i \cdot k^t}$ indexed by $(i_0, i_1, \ldots, i_t) \in [\ell_i] \times [k]^t$, computes a new proof message $\hat{\Pi}_i := \mathrm{Enc}_{1,\ldots,t}(\Pi_i) \in \mathbb{F}^{\ell_i \cdot n^t}$, and sends $\hat{\Pi}_i$ to $\hat{\mathbf{V}}$. Also, $\hat{\mathbf{P}}$ forwards any non-oracle messages from $\mathbf{P}$ to $\mathbf{V}$ via $\hat{\mathbf{V}}$.

2. *Verifier messages.* $\hat{\mathbf{V}}$ receives challenge message ρ_i from $\mathbf{V}$ and forwards it to $\hat{\mathbf{P}}$, who forwards it to $\mathbf{P}$.

3. *Tensor queries.* If $\hat{\mathbf{V}}$ receives any tensor query (or queries) $q \in \mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ on $(\Pi_0, \Pi_1, \ldots, \Pi_i)$ from $\mathbf{V}$, it sends the query q to $\hat{\mathbf{P}}$, who responds by computing $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i)$ themselves and sending it to $\hat{\mathbf{V}}$ as a non-oracle message. Then $\hat{\mathbf{V}}$ forwards this (alleged) query answer to $\mathbf{V}$.

This completes the simulation of the tensor IOP. If at this point the tensor IOP verifier $\mathbf{V}$ rejects, then the IOP verifier $\hat{\mathbf{V}}$ rejects too. (There is no need to check if the IOP prover $\hat{\mathbf{P}}$ answered tensor queries honestly.)

Consistency phase. In this phase the IOP verifier $\hat{\mathbf{V}}$ checks that the IOP prover $\hat{\mathbf{P}}$ honestly answered the tensor queries of the tensor IOP verifier $\mathbf{V}$ in the simulation phase. Suppose that the tensor queries of $\mathbf{V}$ are given by $q^{(s)} = (q_0^{(s)}, q_1^{(s)}, \ldots, q_t^{(s)})$ for each $s \in [\mathsf{q}]$. They are known to $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$. They interact as follows.

1. *Send codewords.* The prover $\hat{\mathbf{P}}$ sends proof messages $\{c_r^{(1)}, \ldots, c_r^{(\mathsf{q})} \in \mathbb{F}^{k \cdot n^{t-r}}\}_{r \in [t]}$ that are computed as described below. (Among these, $c_t^{(1)}, \ldots, c_t^{(\mathsf{q})} \in \mathbb{F}^k$ are sent as non-oracle messages.)

   - First, define the functions

   $$f_0^{(0)} := \mathrm{Stack}\,(\Pi_0, \ldots, \Pi_{\mathsf{rc}}) \in \mathbb{F}^{\ell \cdot k^t} \quad \text{and} \quad c_0^{(0)}(i_0, j_1, \ldots, j_t) := \mathrm{Stack}\left(\hat{\Pi}_0, \ldots, \hat{\Pi}_{\mathsf{rc}}\right) \in \mathbb{F}^{\ell \cdot n^t} \ .$$

   Note that $c_0^{(0)} = \mathrm{Enc}_{1,\ldots,t}(f_0^{(0)})$. Moreover, $\hat{\mathbf{P}}$ already knows the value of $f_0^{(0)}$ and of $c_0^{(0)}$ at every point, and $\hat{\mathbf{V}}$ has point-query access to every value of $c_0^{(0)}$ from the index or the messages sent during the simulation phase.

   - Next, for each $r \in [t]$ and $s \in [\mathsf{q}]$, $\hat{\mathbf{P}}$ computes the following message and its encoding:

   $$f_r^{(s)} := \mathrm{Fold}_{r-1}(f_{r-1}^{(s)}; q_{r-1}^{(s)}) \in \mathbb{F}^{k \cdot k^{t-r}} \quad \text{and} \quad c_r^{(s)} := \mathrm{Enc}_{r+1,\ldots,t}(f_r^{(s)}) \in \mathbb{F}^{k \cdot n^{t-r}} \ .$$

   For $r = 1$, $\mathrm{Fold}_{r-1}(f_{r-1}^{(0)}; q_{r-1}^{(s)})$ is used. When $r = t$, the expression $\mathrm{Enc}_{r+1,\ldots,t}$ is degenerate and no encoding takes place, and so $c_t^{(1)}, \ldots, c_t^{(\mathsf{q})}$ are vectors in $\mathbb{F}^k$.

2. *Proximity test.* The prover $\hat{\mathbf{P}}$ and verifier $\hat{\mathbf{V}}$ engage in the IOP of proximity $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$ for $R_{\otimes}(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$ with index $\mathbb{i} = \bot$, instance $\mathbb{x} = (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$, and witness $\mathbb{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ to show that $c_0^{(0)} \in (\mathcal{C}^{\otimes t})^\ell$ (or at least close) and that $c_r^{(s)} \in (\mathcal{C}^{\otimes t-r})^k$ (or at least close) for all $r \in [t]$ and $s \in [\mathsf{q}]$.[16] (If $\mathbf{V}'$ rejects in this sub-protocol, then $\hat{\mathbf{V}}$ rejects.)

3. *Consistency checks.* The verifier $\hat{\mathbf{V}}$ samples λ tuples of the form $(j_1, \ldots, j_t) \leftarrow [n]^t$ and, for each tuple $(j_1, \ldots, j_t)$, proceeds as follows. For each $r \in [t]$, each $s \in [\mathsf{q}]$, and each $i_r \in [k]$, the verifier $\hat{\mathbf{V}}$ queries the function $c_r^{(s)} \colon [k] \times [n]^{t-r} \to \mathbb{F}$ at $(i_r, j_{r+1}, \ldots, j_t)$. Then, for each $r \in [t]$ and $s \in [\mathsf{q}]$, $\hat{\mathbf{V}}$ checks the following equation:

   $$\mathrm{Fold}_{r-1}(c_{r-1}^{(s)}; q_{r-1}^{(s)})(j_r, \ldots, j_t) = \mathrm{Enc}_r(c_r^{(s)})(j_r, \ldots, j_t) \ .$$

   Finally, for each $s[\mathsf{q}]$, $\hat{\mathbf{V}}$ computes $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$, and checks that it is equal to the answer to the s-th tensor query $q^{(s)} = (q_0^{(s)}, q_1^{(s)}, \ldots, q_t^{(s)})$ that was reported by $\hat{\mathbf{P}}$ in the simulation phase.

[16] The proximity test is invoked on $c_0^{(0)}$ which includes $\hat{\Pi}_0$. This is merely for notational convenience, as $\hat{\Pi}_0 \in \mathcal{C}^{\ell_0}$ was computed by the indexer and is already known to be a valid interleaved codeword.

9.2 Proof of Lemma 9.1

Lemma 9.5 (completeness)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *is perfectly complete.*

Proof. Observe that in the simulation phase, $\hat{\mathbf{P}}$ will compute the correct answers to the tensor queries made by $\mathbf{V}$. The completeness of $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ ensures that $\mathbf{V}$ will accept.

Now we address the consistency phase. Firstly, note that the verifier can compute the verification equations. Queries of $c_r^{(s)}$ at $(i_r, j_{r+1}, \ldots, j_t)$ for each $i_r \in [k]$ are sufficient for computing the values of $\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)})$ and $\mathrm{Enc}_r(c_r^{(s)})$ at $(j_{r+1}, \ldots, j_t)$ because the folding and encoding operations sum over function values at $(i_r, j_{r+1}, \ldots, j_t)$ for each $i_r \in [k]$.

Secondly, note that $\hat{\mathbf{I}}$ and $\hat{\mathbf{P}}$ will always compute encodings honestly, so that $c_0^{(0)} \in (\mathcal{C}^{\otimes t})^\ell$, and for all $r \in [t]$ and $s \in [\mathsf{q}]$, we have $c_r^{(s)} \in (\mathcal{C}^{\otimes t-r})^k$. Thus, the completeness of $(\mathbf{P}', \mathbf{V}')$ ensures that $\mathbf{V}'$ will accept.

The verifier checks the following verification equations pertaining to messages $c_r^{(s)}$ sent during the consistency phase:

$$\mathrm{Fold}_{r-1}(c_{r-1}^{(s)}; q_{r-1}^{(s)}) = \mathrm{Enc}_r(c_r^{(s)}) \qquad \text{for each } r \in [t] \text{ and } s \in [\mathsf{q}]. \tag{21}$$

Setting $(r, f_{r-1}, f_r, c_{r-1}, c_r, q, a) = (r, f_{r-1}^{(s)}, f_r^{(s)}, c_{r-1}^{(s)}, c_r^{(s)}, q_{r-1}^{(s)}, k)$ in Lemma 7.8 gives Equation (21), setting $a = \ell$ instead of k for the case $r = 1$. In each case, the fact that the prover is honest guarantees that Equation (20) from Lemma 7.8 holds.

Finally, consider the verification checks that $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$ is equal to the s-th tensor query $(q_0^{(s)}, \ldots, q_t^{(s)})$. Recall that for all $r \in [t]$ and $s \in [\mathsf{q}]$, we have $f_r^{(s)} = \mathrm{Fold}_{r-1}(f_{r-1}^{(s)}; q_{r-1}^{(s)})$. Further, for all $s \in [\mathsf{q}]$, we have $c_t^{(s)} = f_t^{(s)}$. Expanding using the definition of Fold, we see that

$$\begin{aligned}
\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)}) &= \sum_{i_t \in [k]} q_t^{(s)}(i_t) f_t^{(s)}(i_t) = \sum_{i_t \in [k]} q_t^{(s)}(i_t) \mathrm{Fold}_{t-1}(c_{t-1}^{(s)}; q_{t-1}^{(s)})(i_t) \\
&= \sum_{i_{t-1}, i_t \in [k]} q_{t-1}^{(s)}(i_{t-1}) q_t^{(s)}(i_t) f_{t-1}^{(s)}(i_{t-1}, i_t) \\
&\quad \vdots \\
&= \sum_{i_0 \in [\ell], i_1, \ldots, i_t \in [k]} q_0^{(s)}(i_0) \ldots q_t^{(s)}(i_t) f_0^{(s)}(i_0, \ldots, i_t) \ .
\end{aligned}$$

By definition, $f_0^{(0)} = \mathrm{Stack}\,(\Pi_0, \ldots, \Pi_{\mathsf{rc}})$. Hence $\langle q_0^{(s)} \otimes \ldots \otimes q_t^{(s)}, \mathrm{Stack}\,(\Pi_0, \ldots, \Pi_{\mathsf{rc}}) \rangle = \mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$. □

Lemma 9.6 (soundness)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has soundness error*

$$\epsilon + \epsilon'(1/4) + \left(1 - \frac{\delta^t}{2}\right)^\lambda \ .$$

Proof. Suppose that $(\mathtt{i}, \mathtt{x}) \notin L(R)$, and fix a malicious prover $\hat{\mathbf{P}}$. Let $\hat{\Pi}_0$ be the encoded index and let $\hat{\Pi}_1, \ldots, \hat{\Pi}_{\mathsf{rc}}$ be the oracle messages sent to $\hat{\mathbf{V}}$ during the simulation phase. These define a word $c_0^{(0)} \in \mathbb{F}^{\ell \cdot n^t}$ which, in an honest proof, would be ℓ interleaved $\mathcal{C}^{\otimes t}$-codewords. Later oracle messages sent by $\hat{\mathbf{P}}$ define words $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$ for each $r \in [t]$ and $s \in [\mathsf{q}]$. In an honest proof, $c_r^{(s)}$ would be k interleaved $\mathcal{C}^{\otimes t-r}$-codewords. For each $r \in \{0, 1, \ldots, t-1\}$, let $\Delta_r := \Delta(\{c_r^{(s)}\}_s, \mathcal{C}^{\otimes t-r})$. If there exists $r \in \{0, 1, \ldots, t-1\}$

such that $\Delta_r < \delta^{t-r}/2$, then for all $s \in [\mathsf{q}]$ the function $c_r^{(s)}$ has a unique closest interleaved $\mathcal{C}^{\otimes t-r}$-codeword which we denote by $\overline{c}_r^{(s)}$. We separate the soundness proof into different cases.

- *Case 1:* At least one of the proof messages is far from being a codeword. Formally, there exists $r \in \{0, 1, \ldots, t-1\}$ such that $\Delta_r \geq \delta^{t-r}/4$. By soundness of the proximity test with $\Delta_\otimes = 1/4$, it follows that $(\mathbf{P}', \mathbf{V}')$, $\hat{\mathbf{V}}$ will accept with probability at most $\epsilon'(1/4)$.

- *Case 2:* The proof messages are close to being codewords but the decodings of words $c_r^{(s)}$ from the consistency phase are not consistent across values of r. Formally, for all $r \in \{0, 1, \ldots, t-1\}$ we have $\Delta_r < \delta^{t-r}/4$, but there exists $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$ such that $\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})$.

  By definition of Δ_r, we know that $\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)})$ has relative distance at most Δ_r from $\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)})$.

  We know that $c_{r+1}^{(s)}$ and $\overline{c}_{r+1}^{(s)}$ have blockwise relative distance Δ_{r+1}. Therefore, it is also true that $\mathrm{Enc}_{r+1}(c_{r+1}^{(s)})$ and $\mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})$ have relative distance at most Δ_{r+1}.

  However, $\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)})$ and $\mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})$ are both codewords in $\mathcal{C}^{\otimes t-r}$. Since they are not equal, they must have relative distance at least δ^{t-r}.

  Therefore, we have

$$\begin{aligned}
\delta^{t-r} &\leq \Delta(\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}), \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})) \\
&\leq \Delta(\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}), \mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)})) + \Delta(\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)}), \mathrm{Enc}_{r+1}(c_{r+1}^{(s)})) \\
&\quad + \Delta(\mathrm{Enc}_{r+1}(c_{r+1}^{(s)}), \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})) \\
&\leq \Delta_r + \Delta(\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)}), \mathrm{Enc}_{r+1}(c_{r+1}^{(s)})) + \Delta_{r+1} \\
&\leq \Delta(\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)}), \mathrm{Enc}_{r+1}(c_{r+1}^{(s)})) + \delta^{t-r}/2 \ .
\end{aligned}$$

  This implies that $\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)})$ and $\mathrm{Enc}_{r+1}(c_{r+1}^{(s)})$ differ in at least a $\delta^{t-r}/2$-fraction of possible entries, which is at least a a $\delta^t/2$-fraction. The verifier $\hat{\mathbf{V}}$ checks that $\mathrm{Fold}_r(c_r^{(s)}; q_r^{(s)})$ and $\mathrm{Enc}_{r+1}(c_{r+1}^{(s)})$ are equal at λ random entries. Therefore, $\hat{\mathbf{V}}$ will accept with probability at most $\left(1 - \delta^t/2\right)^\lambda$.

- *Case 3:* Every message is close to being a codeword, and the decodings of words $c_r^{(s)}$ are all consistent over different values of r. Formally, for all $r \in \{0, 1, \ldots, t-1\}$, we have $\Delta_r < \delta^{t-r}/4$ so in particular, for all $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$, the function $\overline{c}_r^{(s)}$ is well-defined (this is trivially true for $r = t$). Then, the consistency checks are satisfied by the *corrected words* $\overline{c}_r^{(s)}$. Namely, for all $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$,

$$\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}) = \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)}) \ .$$

  In particular, for all $r \in r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$, there exist functions $f_{r+1}^{(s)}$ consisting of k vectors in $\mathbb{F}^{k^{t-r}}$ (or ℓ vectors for $r = 0$) such that $\overline{c}_{r+1}^{(s)} = \mathrm{Enc}_{r+2,\ldots,t}(f_{r+1}^{(s)})$ and $f_{r+1}^{(s)} = \mathrm{Fold}_r(f_r^{(s)}; q_r^{(s)})$. The proof of Lemma 9.5 then shows that $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$ is the response to the s-th tensor query on $f_0^{(0)}$.

  If there exists $s \in [\mathsf{q}]$ such that $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$ is not equal to the s-th tensor-query response provided by $\hat{\mathbf{P}}$, then the verifier will reject, as they perform a check that these values are equal. Otherwise, all of the query responses from $\hat{\mathbf{P}}$ have been consistent with a perfect simulation of a $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ execution. By the soundness of $(\mathbf{I}, \mathbf{P}, \mathbf{V})$, the verifier will accept with probability at most ϵ.

□

Lemma 9.7 (proof length)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has proof length* $\rho^{-t}\mathsf{l} + \mathsf{l}' + O(\mathsf{q} \cdot \rho^{-t+1}k^t)$ *over alphabet* $\mathbb{F}$*, plus an additional* $\mathsf{q} + k\mathsf{q}$ *elements of* $\mathbb{F}$ *sent as non-oracle messages (plus any non-oracle messages from the sub-protocols* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *and* $(\mathbf{P}', \mathbf{V}')$*).*

Proof. In the simulation phase, each proof string Π_i is replaced via its encoding $\hat{\Pi}_i$ using $\mathcal{C}^{\otimes t}$. This increases the total number of field elements across all of these from l to $\rho^{-t}\mathsf{l}$.

Subsequently, in the consistency phase, the prover $\hat{\mathbf{P}}$ sends $\mathsf{q} \cdot k$ codewords of $\mathcal{C}^{\otimes t-r}$ for each $r[t-1]$. One round contributes $\mathsf{q} \cdot kn^{t-r} = \mathsf{q} \cdot k(\rho^{-1}k)^{t-r}$ field elements. The total cost over all rounds is given by a geometric sum, and is $O(\mathsf{q} \cdot \rho^{-t+1}k^t)$.

The prover $\hat{\mathbf{P}}$ sends non-oracle messages $c_t^{(s)}$ for each $s \in [\mathsf{q}]$. Each requires k field elements. The prover $\hat{\mathbf{P}}$ also sends the results of the q tensor queries of $\mathbf{V}$ as non-oracle messages, each of which is one field element.

Finally, as part of the proximity test, the prover $\hat{\mathbf{P}}$ sends l' field elements. □

Lemma 9.8 (query complexity)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has query complexity* $\mathsf{q}' + \lambda \cdot \ell + \lambda \cdot k \cdot \mathsf{q}(t-1)$.

Proof. The verifier $\hat{\mathbf{V}}$ makes q' queries for the proximity test. In addition, for λ tuples $(j_1, \ldots, j_t) \in [n]^t$, the verifier $\hat{\mathbf{V}}$ makes the following queries. Firstly, the verifier $\hat{\mathbf{V}}$ makes point queries for the values of $c_0^{(0)}(i_0, j_1, \ldots, j_t)$ for each $i_0 \in [\ell]$. This means queries at $\ell\lambda$ input points. The verifier also makes point queries for the values of $c_r^{(s)}(i_r, j_{r+1}, \ldots, j_t)$ for each $r \in [t-1]$, each $s[\mathsf{q}]$, and each $i_r[k]$. This means queries at $k\lambda$ input points for each of the $\mathsf{q}(t-1)$ functions $c_r^{(s)}$. Summing gives the result. □

Lemma 9.9 (indexer time)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has indexing time* $\frac{\rho^{-t}-1}{\rho^{-1}-1} \cdot \theta(k) \cdot \mathsf{li}$.

Proof. By Lemma 7.5, the computational overhead for computing a tensor code is $\frac{\rho^{-t}-1}{\rho^{-1}-1} \cdot \theta(k)$. The result follows. □

Lemma 9.10 (prover time)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has prover time* $\mathsf{tp} + \mathsf{tp}' + \frac{\rho^{-t}-1}{\rho^{-1}-1} \cdot \mathsf{l} \cdot \theta(k) + O\left(\mathsf{q} \cdot \frac{\rho^{-t+1}}{\rho^{-1}-1} \cdot k^t\right) \cdot \theta(k) + 2\mathsf{q} \cdot \mathsf{l} + O(\mathsf{q}k^t)$.

Proof. The prover $\hat{\mathbf{P}}$ runs the provers $\mathbf{P}$ and $\mathbf{P}'$, which respectively takes time tp and tp'. Moreover, the prover $\hat{\mathbf{P}}$ must produce the codewords $c_r^{(s)}$ and the answers to the tensor queries. Note that by Remark 9.3, as $\hat{\mathbf{P}}$ has access to the output of the indexer $\mathbf{I}$, the prover $\hat{\mathbf{P}}$ is able to compute the answers to tensor queries involving the index. Otherwise, $\hat{\mathbf{P}}$ would have to run the indexer $\mathbf{I}$ and the running time of $\hat{\mathbf{P}}$ would include the term ti.

First, we focus on the costs of computing the functions $f_r^{(s)}$ and the answers to the tensor queries.

Recall that $f_0^{(0)}$ consists of $\mathbb{F}^{\ell \cdot k^t}$ field elements, which can be viewed as ℓ vectors in $\mathbb{F}^{k^t}$, and $\mathsf{l} = \ell \cdot k^t$. For each $s \in [\mathsf{q}]$, the function $f_1^{(s)}$ is a linear combination of these $\mathbb{F}^{k^t}$ vectors. Computing each linear combination costs $(2\ell - 1)k^t$ operations, for a total of at most total of at most $2\mathsf{q} \cdot \mathsf{l}$ operations.

In subsequent rounds, for each $r \in [t]$ and $s \in [\mathsf{q}]$, the prover $\hat{\mathbf{P}}$ has the function $f_r^{(s)}$, which can be viewed as k vectors in $\mathbb{F}^{k^{t-r}}$. The prover must compute $f_{r+1}^{(s)}$ for each $s \in [\mathsf{q}]$. Each $f_{r+1}^{(s)}$ is just computed from $f_r^{(s)}$ and is a linear combination of k vectors, using $\mathsf{q}(2k-1)k^{t-r}$ operations in total. The total cost over all rounds is given by a geometric sum, and is $O(\mathsf{q}k^t)$ operations.

Now we focus on the costs of encoding each function $f_r^{(s)}$ to get the codeword $c_r^{(s)}$.

The first message $f_0^{(0)}$ consists of ℓ vectors in $\mathbb{F}^{k^t}$. The cost of encoding one of these ℓ vectors to get a codeword in $\mathcal{C}^{\otimes t}$ is $\frac{\rho^{-t}-1}{\rho^{-1}-1} \cdot k^t \cdot \theta(k)$ operations by Lemma 7.5.

In subsequent rounds, for each $r \in [t-1]$ and $s \in [\mathsf{q}]$, the prover has functions $f_r^{(s)}$, each of which consists of k vectors in $\mathbb{F}^{k^{t-r}}$ as previously discussed. By Lemma 7.5 it costs $\frac{\rho^{-t+r}-1}{\rho^{-1}} \cdot k^{t-r} \cdot \theta(k)$ operations to encode each vector into a $\mathcal{C}^{\otimes t-r}$-codeword, giving $\mathsf{q} \cdot \frac{\rho^{-t+r}-1}{\rho^{-1}-1} \cdot k^{t-r+1} \cdot \theta(k)$ arithmetic operations. The total cost over all rounds is given by a geometric sum, and is $O(\mathsf{q} \cdot \frac{\rho^{-t+1}}{\rho^{-1}-1} \cdot k^t) \cdot \theta(k)$.

Summing all of these contributions gives the stated prover time. □

Lemma 9.11 (verifier time)**.** $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has verifier time* $\mathsf{tv} + \mathsf{tv}' + \lambda \cdot \mathsf{q} \cdot O(\ell + t \cdot \theta(k) \cdot k)$.

Proof. The verifier $\hat{\mathbf{V}}$ runs the provers $\mathbf{V}$ and $\mathbf{V}'$, which respectively takes time tv and tv'.

Moreover, the verifier $\hat{\mathbf{V}}$ computes verification equations for λ tuples $(j_1, \ldots, j_t) \in [n]^t$. Consider the computations for one of those tuples.

Computing $\mathrm{Fold}(c_r^{(s)}; \alpha)(j_{r+1}, \ldots, j_t) = \sum_{i_r \in [k]} \alpha(i) c_r^{(s)}(i, j_{r+1}, \ldots, j_t)$ from query responses is like computing a scalar product of vectors of length k, which requires $2k - 1 = O(k)$ arithmetic operations. The verifier computes q foldings of the vector of ℓ values from $c_0^{(0)}$, and a folding of $c_r^{(s)}$ for each $r \in [t]$ and $s \in [\mathsf{q}]$. This gives $O(\mathsf{q}\ell + \mathsf{q}tk)$ arithmetic operations for computing the values of folded functions.

The verifier $\hat{\mathbf{V}}$ also has to compute $\mathrm{Enc}_r(c_r^{(s)})(j_r, \ldots, j_t)$ for each $r \in [t]$ and $s \in [\mathsf{q}]$. An encoding of a vector of length k costs θk operations. This gives $\mathsf{q} \cdot t \cdot \theta(k) \cdot k$ operations for computing encodings.

Summing all of these contributions gives the stated verifier time. □

10 Proximity test for tensor codes

Lemma 10.1. *Let $\mathcal{C}$ be a linear code over $\mathbb{F}$ with rate $\rho = k/n$, relative distance $\delta = d/n$, and encoding arithmetic complexity θk. There exists a interactive oracle proof of proximity $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$ with point queries for the relation $R_{\otimes}(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$ with the following parameters:*

- *soundness error $\epsilon'(\Delta_{\otimes}) = O(d^t/|\mathbb{F}|) + \left(1 - \min\{\delta^t/4, \Delta_{\otimes}\}\right)^{\lambda}$;*
- *round complexity t;*
- *proof length $O(n^t)$;*
- *query complexity $O(\lambda \cdot (\ell + kt \cdot \mathsf{q}))$;*
- *prover time $O_{\rho,t}((\ell + \mathsf{q}) \cdot k^t) \cdot \theta(k)$; and*
- *verifier time $O(\lambda \cdot (\ell + \theta(k) \cdot kt \cdot \mathsf{q}))$.*

Remark 10.2. The stated time for the (honest) prover $\mathbf{P}'$ requires having access to the un-encoded messages $f_r^{(s)} \in \mathbb{F}^{k \cdot k^{t-r}}$ corresponding to the alleged codewords $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$ for each $r \in [t-1]$ and $s \in [\mathsf{q}]$ (and $f_0^{(0)} \in \mathbb{F}^{\ell \cdot k^t}$ corresponding to $c_0^{(0)} \in \mathbb{F}^{\ell \cdot n^t}$). No other properties of the proximity test depend on this.

10.1 Construction

Construction 10.3. We describe the construction of $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$. The prover $\mathbf{P}'$ takes as input an index $\mathtt{i} = \perp$, instance $\mathtt{x} = (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$, and witness $\mathtt{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$, while the verifier $\mathbf{V}'$ takes as input the index $\mathtt{i}$ and the instance $\mathtt{x}$.

1. *Interactive phase.* For each round $r \in [t]$:
   - $\mathbf{V}'$ sends random challenge messages $\alpha_{r-1}^{(0)}, \ldots, \alpha_{r-1}^{(\mathsf{q})} \in \mathbb{F}^k$. (For $r = 1$, $\mathbf{V}'$ sends only $\alpha_0^{(0)} \in \mathbb{F}^{\ell}$.)
   - $\mathbf{P}'$ sends the proof message $c_r^{(0)} \in \mathbb{F}^{k \cdot n^{t-r}}$ computed as

$$f_r^{(0)} := \mathrm{Fold}_{r-1}(\{f_{r-1}^{(s)}\}_s; \{\alpha_{r-1}^{(s)}\}_s) \quad \text{and} \quad c_r^{(0)} := \mathrm{Enc}_{r+1,\ldots,t}(f_r^{(0)}) \ .$$

   (For $r = 1$, $\mathrm{Fold}_{r-1}(\{f_{r-1}^{(s)}\}_s; \{\alpha_{r-1}^{(s)}\}_s)$ is just over $f_0^{(0)}$ and $\alpha_0^{(0)}$.)

   Note that when $r = t$ the expression $\mathrm{Enc}_{r+1,\ldots,t}$ is degenerate and no encoding takes place. Moreover, in this final round, $\mathbf{P}'$ sends $c_t^{(0)} \in \mathbb{F}^k$ as a non-oracle message (as the verifier $\mathbf{V}'$ will read all of $c_t^{(0)}$).

2. *Query phase.* The verifier $\mathbf{V}'$ samples λ tuples of the form $(j_1, \ldots, j_t) \leftarrow [n]^t$ and proceeds as follows for each tuple. For each $s \in \{0, 1, \ldots, \mathsf{q}\}$, $\mathbf{V}'$ queries the function $c_r^{(s)} \colon [k] \times [n]^{t-r} \to \mathbb{F}$ at $(i_r, j_{r+1}, \ldots, j_t)$ for each $i_r \in [k]$ and checks, for each $r \in [t]$, the following equation:

$$\mathrm{Fold}_{r-1}(\{c_{r-1}^{(s)}\}_s; \{\alpha_{r-1}^{(s)}\}_s)(j_r, \ldots, j_t) = \mathrm{Enc}_r(c_r^{(0)})(j_r, \ldots, j_t) \ . \tag{22}$$

10.2 Proof of Lemma 10.1

Lemma 10.4 (completeness)**.** $(\mathbf{P}', \mathbf{V}')$ *has perfect completeness.*

Proof. We begin by noting that the queries made by $\mathbf{V}'$ suffice to perform the checks in its query phase (see Equation (22)). Indeed, queries to $c_r^{(s)}$ at $(i_r, j_{r+1}, \ldots, j_t)$ for each $i_r \in [k]$ suffice to compute the values of $\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$ and $\mathrm{Enc}_r(c_r^{(0)})$ at $(j_{r+1}, \ldots, j_t)$ because the folding and encoding operations sum over function values at $(i_r, j_{r+1}, \ldots, j_t)$ for each $i_r \in [k]$.

Next, observe that the verifier $\mathbf{V}'$ checks, for each $r \in [t]$, the following equation:

$$\mathrm{Fold}_{r-1}(\{c_{r-1}^{(s)}\}_s; \{\alpha_{r-1}^{(s)}\}_s) = \mathrm{Enc}_r(c_r^{(0)}) \ . \tag{23}$$

Setting

$$(r, f_{r-1}, f_r, c_{r-1}, c_r, q, a) := \left(r, f_{r-1}^{(s)}, \mathrm{Fold}_{r-1}(f_{r-1}^{(s)}; \alpha_{r-1}^{(s)}), c_{r-1}^{(s)}, \mathrm{Fold}_{r-1}(c_{r-1}^{(s)}; \alpha_{r-1}^{(s)}), \alpha_{r-1}^{(s)}, k\right)$$

in Lemma 7.8 shows that

$$\mathrm{Fold}_{r-1}(\mathrm{Enc}_{r,\ldots,t}(f_{r-1}^{(s)}); \alpha_{r-1}^{(s)}) = \mathrm{Enc}_{r,\ldots,t}(\mathrm{Fold}_{r-1}(f_{r-1}^{(s)}; \alpha_{r-1}^{(s)})) \ .$$

Summing for $s \in \{0, 1, \ldots, \mathsf{q}\}$ and using the linearity of $\mathrm{Enc}_{r,\ldots,t}$ gives Equation (23). Again, set $a := \ell$ instead of k for the case $r = 1$.

In each case, the honest prover $\mathbf{P}'$ guarantees that Equation (20) from Lemma 7.8 holds. □

Lemma 10.5 (soundness)**.** $(\mathbf{P}', \mathbf{V}')$ *has soundness error*

$$O\left(\frac{d^t}{|\mathbb{F}|}\right) + \left(1 - \min\{\delta^t/4, \Delta_\otimes\}\right)^\lambda \ .$$

The proof of Lemma 10.5 is given in Section 10.3.

Lemma 10.6 (proof length)**.** $(\mathbf{P}', \mathbf{V}')$ *has proof length* $O(n^t)$ *over alphabet* $\mathbb{F}$*, plus an additional* k *elements of* $\mathbb{F}$ *sent as non-oracle messages.*

Proof. In the interactive phase, the prover $\mathbf{P}'$ sends k codewords of $\mathcal{C}^{\otimes t-r}$ for each $r \in [t-1]$. This means that the r-th round contributes $k \cdot n^{t-r}$ field elements. The total number across all rounds is, via a geometric sum, $O(n^t)$. In addition, the prover $\mathbf{P}'$ sends the non-oracle message $c_t^{(0)}$, which consists of k field elements. □

Lemma 10.7 (query complexity)**.** $(\mathbf{P}', \mathbf{V}')$ *has query complexity* $\lambda \cdot O(\ell + kt \cdot \mathsf{q})$.

Proof. For λ tuples $(j_1, \ldots, j_t) \in [n]^t$, the verifier $\mathbf{V}'$ makes the following queries. Firstly, the verifier $\mathbf{V}'$ makes point queries for the values of $c_0^{(0)}(i_0, j_1, \ldots, j_t)$ for each $i_0 \in [\ell]$. This means queries at $\ell\lambda$ input points. The verifier also makes point queries for the values of $c_r^{(s)}(i_r, j_{r+1}, \ldots, j_t)$ for each $r \in [t-1]$, each $s \in \{0, 1, \ldots, \mathsf{q}\}$, and each $i_r \in [k]$. This means queries at $k\lambda$ input points for each of the $(\mathsf{q}+1)(t-1)$ functions $c_r^{(s)}$. Summing gives the result. □

Lemma 10.8 (prover time)**.** $(\mathbf{P}', \mathbf{V}')$ *has prover time* $O\left(\frac{\rho^{-t+1}}{\rho^{-1}-1} \cdot k^t\right) \cdot \theta(k) + 2\ell k^t + O(\mathsf{q}k^t)$.

Proof. The prover $\mathbf{P}'$ must produce, for each $r \in [t]$, the function $c_r^{(0)}$ which consists of k interleaved $\mathcal{C}^{\otimes t-r}$-codewords. We compute the cost by first focusing on the costs of computing $f_r^{(0)}$ for each $r \in [t]$, and then considering the costs of encoding $f_r^{(0)}$ to get $c_r^{(0)}$.

Recall that $f_0^{(0)}$ consists of $\mathbb{F}^{\ell \cdot k^t}$ field elements, which can be viewed as ℓ vectors in $\mathbb{F}^{k^t}$. The function $f_1^{(0)}$ is a linear combination of these $\mathbb{F}^{k^t}$ vectors which costs $(2\ell - 1)k^t \leq 2\ell k^t$ operations to compute.

In subsequent rounds, for each $r \in [t]$, the prover $\mathbf{P}'$ has functions $\{f_r^{(s)}\}_{s \in \{0,1,\ldots,\mathsf{q}\}}$, each of which can be viewed as k vectors in $\mathbb{F}^{k^{t-r}}$. The prover must compute $f_{r+1}^{(0)} = \mathrm{Fold}_r(\{f_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$, a linear

combination of $(\mathsf{q}+1)k$ vectors, which costs $(2(\mathsf{q}+1)k-1)k^{t-r}$ operations. The total cost over all rounds is given by a geometric sum, and is $O(\mathsf{q}k^t)$ operations.

Now we focus on the costs of encoding each $f_r^{(0)}$ to get $c_r^{(0)}$. For each $r \in [t-1]$, the prover $\mathbf{P}'$ has the function $f_r^{(0)}$, which can be viewed as k vectors in $\mathbb{F}^{k^{t-r}}$ as previously discussed. By Lemma 7.5 it costs $\frac{\rho^{-t+r}-1}{\rho^{-1}} \cdot k^{t-r} \cdot \theta(k)$ operations to encode each vector into a $\mathcal{C}^{\otimes t-r}$ codeword, giving $\frac{\rho^{-t+r}-1}{\rho^{-1}-1} \cdot k^{t-r+1} \cdot \theta(k)$ operations. The total cost over all rounds is given by a geometric sum, and is $O\left(\frac{\rho^{-t+1}}{\rho^{-1}-1} \cdot k^t\right) \cdot \theta(k)$.

Summing all of these contributions gives the claimed prover time. □

Lemma 10.9 (verifier time)**.** $(\mathbf{P}', \mathbf{V}')$ *has verifier time* $\lambda \cdot O(\ell + t \cdot \mathsf{q}k + t \cdot \theta(k) \cdot k)$.

Proof. The verifier $\mathbf{V}'$ performs checks for λ tuples $(j_1, \ldots, j_t) \in [n]^t$. Consider the checks for one of those tuples.

Computing the value of $\mathrm{Fold}(c_0^{(0)}; \alpha_0^{(0)})$ at $(j_1, \ldots, j_t)$ is like computing a scalar product of vectors of length ℓ, which requires $2\ell - 1 = O(\ell)$ operations. Similarly, for each $r \in [t]$, computing the value of $\mathrm{Fold}(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s) = \sum_s \mathrm{Fold}(c^{(s)}; \alpha^{(s)})$ at $(j_{r+1}, \ldots, j_t)$ is like computing a scalar product of vectors of length $(\mathsf{q}+1)k$, which requires $2(\mathsf{q}+1)k - 1 = O(\mathsf{q} \cdot k)$ operations.

Moreover, for each $r \in [t]$, the verifier $\mathbf{V}'$ has to compute $\mathrm{Enc}_r(c_r^{(0)})(j_r, \ldots, j_t)$. An encoding of a vector of length k costs $\theta(k) \cdot k$ operations. This gives $t \cdot \theta(k) \cdot k$ operations for computing encodings.

The result follows by summing these contributions and multiplying by λ. □

10.3 Soundness of proximity test

We prove that $(\mathbf{P}', \mathbf{V}')$ has soundness error

$$\frac{d(d^t-1)}{4(d-1)|\mathbb{F}|} + \left(1 - \min\{\delta^t/4, \Delta_\otimes\}\right)^\lambda \quad .$$

Suppose that $(\mathtt{i}, \mathtt{x}) \notin L(R_\otimes)$, and fix a malicious prover $\mathbf{P}'$. The witness $\mathtt{w}$ and the oracle messages sent by $\mathbf{P}'$ define functions $c_r^{(s)}$ for $r \in [t]$ and $s \in [\mathsf{q}]$. If $\mathbf{P}'$ were honest, $c_r^{(s)}$ would be a codeword in $(\mathcal{C}^{\otimes t-r})^k$, except for $c_0^{(0)}$ which would be a codeword in $(\mathcal{C}^{\otimes t})^\ell$.

10.3.1 Definitions

We define the sets Fail_r, which contain the tuples $(j_r, \ldots, j_t)$ for which the consistency checks between the functions $c_{r-1}^{(s)}$ and $c_r^{(s)}$ are not satisfied:

$$\mathrm{Fail}_r := \left\{ (j_r, \ldots, j_t) \in [n]^{t-r+1} \middle| \mathrm{Fold}_{r-1}(\{c_{r-1}^{(s)}\}_s; \{\alpha_{r-1}^{(s)}\}_s)(j_r, \ldots, j_t) \neq \mathrm{Enc}_r(c_r^{(0)})(j_r, \ldots, j_t) \right\} \quad .$$

Let $\Delta_r := \Delta(\{c_r^{(s)}\}_s, \mathcal{C}^{\otimes t-r})$ for each $r \in \{0, 1, \ldots, t-1\}$. Let $d_r := \Delta_r \cdot n^{t-r}$ be the blockwise Hamming distance.

If there exists $r \in \{0, 1, \ldots, t\}$ such that $\Delta_r < \delta^{t-r}/2$, then for all $s \in [\mathsf{q}]$ the function $c_r^{(s)}$ has a unique closest codeword in $(\mathcal{C}^{\otimes t-r})^k$ which we denote by $\overline{c}_r^{(s)}$. For each $r \in \{0, 1, \ldots, t-1\}$, we also define the

set Err_r, which contains the tuples $(j_r, \ldots, j_t)$ for which there exists $s \in [\mathsf{q}]$ such that $c_r^{(s)} \neq \overline{c}_r^{(s)}$, so that $(j_r, \ldots, j_t)$ contributes to Δ_r:

$$\mathrm{Err}_r := \left\{ (j_{r+1}, \ldots, j_t) \in [n]^{t-r} \middle| \exists (s, i_r) \ : \ c^{(s)}(i_r, j_{r+1}, \ldots, j_t) \neq \overline{c}^{(s)}(i_r, j_{r+1}, \ldots, j_t) \right\} \ .$$

Note that for $r = t$, the definitions of Δ_t and Err_t are degenerate. The function $c_t^{(0)}$ is not an alleged codeword from any code, but simply an element of $\mathbb{F}^k$. For this reason, we set $\Delta_t = 0$, $\mathrm{Err}_t = \emptyset$ and $\overline{c}_t^{(0)} = c_t^{(0)}$.

We define the drop-set of the set of functions $\{c_r^{(s)}\}_s$.

$$\mathrm{Drop}(\{c^{(s)}\}_s, \gamma) := \left\{ \{\alpha_r^{(s)}\}_s \ \middle| \ \Delta(\mathrm{Fold}(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s), \mathcal{C}^{\otimes t-r}) < \gamma \right\} \ .$$

For each $r \in \{0, 1, \ldots, t\}$, we define $\Delta_r^\dagger := \min\{\delta^{t-r}/4, \Delta_\otimes\}$. Finally, we define distortion sets:

$$\mathrm{Dist}_r := \left\{ \{\alpha_r^{(s)}\}_s \ \middle| \ \{\alpha_r^{(s)}\}_s \in \mathrm{Drop}(\{c_r^{(s)}\}_s, \min\{\Delta_r, \Delta_r^\dagger\}) \right\} \ .$$

10.3.2 Proof cases

We separate the soundness proof into two cases.

- *Case 1:* Distortion occurs. Formally, there exists $r \in \{0, 1, \ldots, t-1\}$ such that $\{\alpha_r^{(s)}\}_s \in \mathrm{Dist}_r$. In Section 10.3.3, we argue that this occurs with probability at most $\frac{d(d^t-1)}{4(d-1)|\mathbb{F}|}$.
- *Case 2:* No distortion occurs. At least one of the proof messages is far from being a codeword, or the proof messages are close to being codewords but the decoded messages from different rounds of the interactive phase are not consistent. Formally, for all $r \in \{0, 1, \ldots, t-1\}$, we have $\{\alpha_r^{(s)}\}_s \notin \mathrm{Dist}_r$ but there exists $r \in \{0, 1, \ldots, t-1\}$ such that one of the following conditions is satisfied:
  - $\Delta_r \geq \Delta_r^\dagger$,
  - $\Delta_r < \Delta_r^\dagger$ and $\mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(0)})$.

  In Section 10.3.4, we argue that $\mathbf{V}'$ accepts with probability at most $\left(1 - \min\{\delta^t/4, \Delta_\otimes\}\right)^\lambda$.

10.3.3 Case 1

Let $r \in \{0, \ldots, t-1\}$. We give an upper bound on the probability that $\{\alpha_r^{(s)}\}_s \in \mathrm{Dist}_r$.

Lemma 10.10 ([AHIV17])**.** *Consider an $[n, k, d]_\mathbb{F}$ code $\mathcal{C}$. Let e be a positive integer such that $e < d/4$. Let $U \in (\mathbb{F}^n)^\ell$ with blockwise-distance $d(U, \mathcal{C}^\ell) > e$. Then, for a random w in the row-span of U, we have*

$$\Pr\left[d(w, \mathcal{C}) \leq e\right] \leq \frac{e+1}{|\mathbb{F}|}$$

For each $r \in [t-1]$ and $s \in [\mathsf{q}]$, the word $c_r^{(s)}$ consists of k interleaved vectors in $\mathbb{F}^{n^{t-r}}$. Notice that for random $\alpha_r^{(s)} \in \mathbb{F}^k$, we have that $\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s) \in \mathbb{F}^{n^{t-r}}$ is a random element of the row-span of all vectors of $c_r^{(s)}$, taking the span over each $s \in [\mathsf{q}]$.

We apply Lemma 10.10 to $\mathcal{C}^{\otimes t-r}$ with $e = n^{t-r} \cdot \min\{\Delta_r, \Delta_r^\dagger\} - 1$, which satisfies $e \leq d^{t-r}/4 - 1$. This shows that $\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$ has relative distance strictly less than $\min\{\Delta_r, \Delta_r^\dagger\}$ with probability at most $d^{t-r}/4|\mathbb{F}|$. The bound $\frac{d(d^t-1)}{4(d-1)|\mathbb{F}|}$ follows by applying a union bound over $r \in \{0, 1, \ldots, t-1\}$ and summing a geometric series.

10.3.4 Case 2

Let $r^* \in \{0, 1, \ldots, t-1\}$ be the largest value of r such that one of the following conditions is satisfied:

- $\Delta_r \geq \Delta_r^\dagger$
- $\Delta_r < \Delta_r^\dagger$ and $\mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(0)})$

Note that if $\Delta_{r^*+1} < \Delta_{r^*+1}^\dagger$ then $\Delta_{r^*+1} \leq \delta^{t-r^*-1}/4$ so for each $s \in \{0, \ldots, \mathsf{q}\}$, $\overline{c}_{r^*+1}^{(s)}$ is uniquely defined, and Err_{r^*+1} is well defined.

Claim 10.11.

$$\Delta_{r^*}^\dagger \leq \frac{|\mathrm{Fail}_{r^*+1} \cup ([n] \times \mathrm{Err}_{r^*+1})|}{n^{t-r^*}} \ .$$

Proof. We will prove the inequality by bounding the right-hand side from below by some auxiliary value, and bounding the left-hand side from above by the same value. The value is used only for the purposes of the proof. Exactly which value we use depends on which condition from Case 2 is satisfied.

Lower bound on the right-hand side. Suppose that $(j_{r^*+1}, \ldots, j_t) \notin \mathrm{Fail}_{r^*+1} \cup ([n] \times \mathrm{Err}_{r^*+1})$.

We have $(j_{r^*+1}, \ldots, j_t) \notin ([n] \times \mathrm{Err}_{r^*+1})$ and so $(j_{r^*+2}, \ldots, j_t) \notin \mathrm{Err}_{r^*+1}$. This means that for all $s \in [\mathsf{q}]$ and $i_{r^*+1} \in [k]$,

$$c_{r^*+1}^{(s)}(i_{r^*+1}, j_{r^*+2}, \ldots, j_t) = \overline{c}_{r^*+1}^{(s)}(i_{r^*+1}, j_{r^*+2}, \ldots, j_t).$$

Applying Enc_{r^*+1} to both sides, we see that for all $s \in [\mathsf{q}]$,

$$\mathrm{Enc}_{r^*+1}(c_{r^*+1}^{(s)})(j_{r^*+1}, \ldots, j_t) = \mathrm{Enc}_{r^*+1}(\overline{c}_{r^*+1}^{(s)})(j_{r^*+1}, \ldots, j_t). \tag{24}$$

We also have $(j_{r^*+1}, \ldots, j_t) \notin \mathrm{Fail}_{r^*+1}$, so

$$\mathrm{Fold}_{r^*}(\{c_{r^*}^{(s)}\}_s; \{\alpha_{r^*}^{(s)}\}_s)(j_{r^*+1}, \ldots, j_t) = \mathrm{Enc}_{r^*+1}(c_{r^*+1}^{(0)})(j_{r^*+1}, \ldots, j_t). \tag{25}$$

Combining Equation (24) with Equation (25) gives

$$\mathrm{Fold}_{r^*}(\{c_{r^*}^{(s)}\}_s; \{\alpha_{r^*}^{(s)}\}_s)(j_{r^*+1}, \ldots, j_t) = \mathrm{Enc}_{r^*+1}(\overline{c}_{r^*+1}^{(0)})(j_{r^*+1}, \ldots, j_t).$$

This implies that $|\mathrm{Fail}_{r^*+1} \cup ([n] \times \mathrm{Err}_{r^*+1})| / n^{t-r^*}$ is greater than or equal to

$$\Delta(\mathrm{Fold}_{r^*}(\{c_{r^*}^{(s)}\}_s; \{\alpha_{r^*}^{(s)}\}_s), \mathrm{Enc}_{r^*+1}(\overline{c}_{r^*+1}^{(0)})).$$

Upper bound on the left-hand side. We give different upper bounds according to which of the two conditions from Case 2 is satisfied.

- $\Delta_r^* \geq \Delta_{r^*}^\dagger$.
  Since $\{c_{r^*}^{(s)}\}_s \notin \mathrm{Dist}_{r^*}$, we know that $\mathrm{Fold}_{r^*}(\{c_{r^*}^{(s)}\}_s; \{\alpha_{r^*}^{(s)}\}_s)$ has relative distance at least $\Delta_r^\dagger$ from $\mathcal{C}^{\otimes t-r^*}$ and hence from $\mathrm{Enc}_{r^*+1}(\overline{c}_{r^*+1}^{(0)})$. This is because $\overline{c}_{r^*+1}^{(0)} \in (\mathcal{C}^{\otimes t-r^*-1})^k$, so $\mathrm{Enc}_{r^*+1}(\overline{c}_{r^*+1}^{(0)}) \in \mathcal{C}^{\otimes t-r^*}$.

- $\Delta^*_r < \Delta^\dagger_{r^*}$ and $\mathrm{Fold}_{r^*}(\{\overline{c}^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s) \neq \mathrm{Enc}_{r^*+1}(\overline{c}^{(0)}_{r^*+1})$.
  Observe that both $\mathrm{Fold}_{r^*}(\{\overline{c}^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s)$ and $\mathrm{Enc}_{r^*+1}(\overline{c}^{(s)}_{r^*+1})$ are $\mathcal{C}^{\otimes t-r^*}$-codewords. Since they are distinct, they have relative distance at least δ^{t-r^*}. Applying the triangle inequality, we learn that

$$\begin{aligned}\delta^{t-r^*} &\leq \Delta(\mathrm{Fold}_{r^*}(\{\overline{c}^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s), \mathrm{Enc}_{r^*+1}(\overline{c}^{(0)}_{r^*+1})) \\ &\leq \Delta(\mathrm{Fold}_{r^*}(\{\overline{c}^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s), \mathrm{Fold}_{r^*}(\{c^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s)) \\ &\quad + \ \Delta(\mathrm{Fold}_{r^*}(\{c^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s), \mathrm{Enc}_{r^*+1}(\overline{c}^{(0)}_{r^*+1})).\end{aligned} \tag{26}$$

  Since $\Delta_{r^*} < \Delta^\dagger_{r^*} \leq \delta^{t-r^*}/4$, the term on the middle line of Equation (26) is at most Δ_{r^*}. We have

$$\Delta(\mathrm{Fold}_{r^*}(\{c^{(s)}_{r^*}\}_s; \{\alpha^{(s)}_{r^*}\}_s), \mathrm{Enc}_{r^*+1}(\overline{c}^{(0)}_{r^*+1})) \geq \delta^{t-r^*} - \Delta_{r^*} > 3\delta^{t-r^*}/4 > \Delta^\dagger_{r^*}.$$

Combining one of the lower bounds with one of the upper bounds depending on which of the two conditions from Case 2 is satisfied implies Claim 10.11. □

Using the result of Claim 10.11, we are ready to give an upper-bound on the prover's success probability in Case 2. Consider a sequence $(j_1, \ldots, j_t) \in [n]^t$. We know that if $(j_{r^*+1}, \ldots, j_t) \in \mathrm{Fail}_{r^*+1}$, then one of the verification equations is not satisfied and $\mathbf{V}'$ will reject.

If $r^*+1 = t$ then since $\mathrm{Err}_t = \emptyset$ by definition, we deduce that $|\mathrm{Fail}_t|/n = |\mathrm{Fail}_t \cup ([n] \times \mathrm{Err}_t)|/n \geq \Delta^\dagger_t$, giving a lower bound on the probability that $\mathbf{V}'$ will reject.

On the other hand, if $r^* + 1 < t$, we cannot assume that Err_{r^*+1} is empty. We want the lower bound of Claim 10.11, which relates to $\mathrm{Fail}_{r^*+1} \cup ([n] \times \mathrm{Err}_{r^*+1})$, to be give a useful bound for the probability that $\mathbf{V}'$ rejects. We have already dealt with Fail_{r^*+1}, but we must show that if $(j_{r^*+1}, \ldots, j_t) \in [n] \times \mathrm{Err}_{r^*+1}$, then the verifier rejects. Note that this condition simplifies to $(j_{r^*+2}, \ldots, j_t) \in \mathrm{Err}_{r^*+1}$.

Since r^* was maximal, we know that the following conditions hold for all $r \in \{r^*, \ldots, t-1\}$.

$$\Delta_r < \Delta^\dagger_r \tag{27}$$

$$\mathrm{Fold}_r(\{\overline{c}^{(s)}_r\}_s; \{\alpha^{(s)}_r\}_s) = \mathrm{Enc}_{r+1}(\overline{c}^{(0)}_{r+1}) \tag{28}$$

In Section 10.3.5, we show that when these conditions hold, then $\mathbf{V}'$ will reject whenever $(j_{r^*+2}, \ldots, j_t) \in \mathrm{Err}_{r^*+1}$. This implies that $\mathbf{V}'$ will reject with probability at least $\Delta^\dagger_{r^*}$. The soundness error follows from observing that in general, $\mathbf{V}'$ rejects with probability at least $\min_r\{\Delta^\dagger_r\} = \min\{\delta/4, \delta^2/4, \ldots, \delta^t/4, \Delta_\otimes\}$ which is equal to $\min\{\delta^t/4, \Delta_\otimes\}$.

10.3.5 Rejecting Err_{r^*+1}

Suppose that $r^* + 1 < t$ and that Equation (27) and Equation (28) hold for all $r \in \{r^*, \ldots, t-1\}$. We show that whenever $(j_{r^*+2}, \ldots, j_t) \in \mathrm{Err}_{r^*+1}$, then $\mathbf{V}'$ will reject. Suppose then, that $(j_{r^*+2}, \ldots, j_t) \in \mathrm{Err}_{r^*+1}$.

Since $\Delta_r < \Delta^\dagger_r = \min\{\delta^{t-r}/4, \Delta_\otimes\}$, we have $\Delta_r < \delta^{t-r}/4$ and each corrected codeword function $\overline{c}^{(s)}_r$ is well-defined.

Let $r \in \{r^*, \ldots, t-1\}$ be maximal such that $(j_{r+1}, \ldots, j_t) \in \mathrm{Err}_r$. We know that such an r exists because $(j_{r^*+2}, \ldots, j_t) \in \mathrm{Err}_{r^*+1}$.

This means that there exists some $i_r \in [k]$ and $s \in [\mathsf{q}]$ such that $c^{(s)}_r(i_r, j_{r+1}, \ldots, j_t) \neq \overline{c}^{(s)}_r(i_r, j_{r+1}, \ldots, j_t))$.

For $(j_{r+1}, \ldots, j_t) \notin \mathrm{Err}_r$, we have $c_r^{(s)}(i_r, j_{r+1}, \ldots, j_t) = \overline{c}_r^{(s)}(i_r, j_{r+1}, \ldots, j_t)$ for all $s \in [\mathsf{q}]$ and $i_r \in [k]$. Hence for $(j_{r+1}, \ldots, j_t) \notin \mathrm{Err}_r$, we know that

$$\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)(j_{r+1}, \ldots, j_t) = \mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)(j_{r+1}, \ldots, j_t).$$

So the foldings of the functions $c_r^{(s)}$ differ from the foldings of the codewords $\overline{c}_r^{(s)}$ in at most a Δ_r-fraction of inputs $(j_{r+1}, \ldots, j_t)$. Since $\Delta_r < \delta^{t-r}/4$, we know that

$$\overline{\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)} = \mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$$

Further, since $\{\alpha_r^{(s)}\}_s \notin \mathrm{Dist}_r$, we know that $\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$ still has relative distance Δ_r from $\mathcal{C}^{\otimes t-r}$, so for all $(j_{r+1}, \ldots, j_t) \in \mathrm{Err}_r$, we know that $\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)$ also fails to be a codeword in the following sense.

$$\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)(j_{r+1}, \ldots, j_t) \neq \mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)(j_{r+1}, \ldots, j_t). \tag{29}$$

By maximality of r, observe that $(j_{r+2}, \ldots, j_t) \notin \mathrm{Err}_{r+1}$. Therefore for all $i_{r+1} \in [k]$,

$$c_{r+1}^{(s)}(i_{r+1}, j_{r+2}, \ldots, j_t) = \overline{c}_{r+1}^{(s)}(i_{r+1}, j_{r+2}, \ldots, j_t).$$

Setting $s = 0$ and applying Enc_{r+1}, we have

$$\mathrm{Enc}_{r+1}(c_{r+1}^{(0)})(j_{r+1}, \ldots, j_t) = \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(0)})(j_{r+1}, \ldots, j_t). \tag{30}$$

Combining Equation (29) and Equation (30) shows that

$$\mathrm{Fold}_r(\{c_r^{(s)}\}_s; \{\alpha_r^{(s)}\}_s)(j_{r+1}, \ldots, j_t) \neq \mathrm{Enc}_{r+1}(c_{r+1}^{(0)})(j_{r+1}, \ldots, j_t).$$

This implies that $(j_{r+1}, \ldots, j_t) \in \mathrm{Fail}_{r+1}$, and so $\mathbf{V}'$ will reject.

A The transposition principle

The transposition principle states that the cost of left-multiplying by a matrix U is closely related to the cost of left-multiplying by the transpose matrix $U^\intercal$. Below we state this principle as studied in [KKB88], where more details about the facts below can be found.

Definition A.1. *A* **linear algorithm** *over a field $\mathbb{F}$ is a triple $A = (V, E, c)$ where (V, E) is a directed acyclic graph with vertex set V and edge set E, and $c\colon E \to \mathbb{F} \setminus \{0\}$ is an edge labeling. We denote by V_{in} the set of source vertices (in-degree is 0) and by V_{out} the set of sink vertices (out-degree is 0).*

A linear algorithm A computes an $\mathbb{F}$-linear function from $\mathbb{F}^{n_{\text{col}}}$ to $\mathbb{F}^{n_{\text{row}}}$, where $n_{\text{col}} := |V_{\text{in}}|$ and $n_{\text{row}} := |V_{\text{out}}|$, that is defined as follows. First, assign to each source vertex $v \in V_{\text{in}}$ a formal variable x_v, which we view as a linear function over the vector of input variables $x := (x_v)_{v \in V_{\text{in}}}$. Next, proceeding in topological order, assign to each non-source vertex $v \in V \setminus V_{\text{in}}$ the linear function $f_v(x) := \sum_{(w,v)\in E} c_{(w,v)} \cdot f_w(x)$, where f_w is the linear function assigned to the vertex w. Finally, the outputs consist of the linear functions assigned to the sink vertices. We denote by U_A the matrix in $\mathbb{F}^{n_{\text{row}} \times n_{\text{col}}}$ representing the $\mathbb{F}$-linear function computed by A, i.e., the matrix such that $f = U_A x$ for output vector $f = (f_v(x))_{v \in V_{\text{out}}}$. Note that U_A is unique up to the ordering of vertices in V_{in} and V_{out}.

Lemma A.2. *If left-multiplication by $U \in \mathbb{F}^{n_{\text{row}} \times n_{\text{col}}}$ is computable via a linear algorithm A using α additions and χ multiplications, then left-multiplication by $U^\intercal \in \mathbb{F}^{n_{\text{col}} \times n_{\text{row}}}$ is computable via a linear algorithm using $\alpha + n_{\text{row}} - n_{\text{col}}$ additions and χ multiplications.*

Proof sketch. Let $A^\intercal$ be the linear algorithm obtained from A by reversing all edges in the graph of A. One can show that $U_{A^\intercal} = (U_A)^\intercal$, and that $A^\intercal$ can be evaluated via $\alpha + n_{\text{row}} - n_{\text{col}}$ additions and χ multiplications. □

Acknowledgements

We are deeply grateful to Sune K. Jakobsen who was instrumental in the early stages of this research project and provided an initial analysis of a compiler from tensor queries to point queries based on tensor codes. We thank Andrea Cerulli for discussions about error correcting codes.

References

[AHIKV17] Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. "Low-Complexity Cryptographic Hash Functions". In: *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*. ITCS '17. 2017, 7:1–7:31.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104.

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

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

[BBBPWM18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BBCGI19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 67–97.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed–Solomon Interactive Oracle Proofs of Proximity". In: *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*. ICALP '18. 2018, 14:1–14:17.

[BCCGP16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 327–357.

[BCGGHJ17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. "Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability". In: *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security*. ASIACRYPT '17. 2017, pp. 336–365.

[BCGJM18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. "Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution". In: *Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '18. 2018, pp. 595–626.

[BCRSVW19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. 2019, pp. 103–128.

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

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

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 677–706.

[BS06] Eli Ben-Sasson and Madhu Sudan. "Robust locally testable codes and products of codes". In: *Random Structures and Algorithms* 28.4 (2006), pp. 387–402.

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

[Ben+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. SP '14. 2014, pp. 459–474.

[CHMMVW20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 738–768.

[CMS17] Alessandro Chiesa, Peter Manohar, and Igor Shinkar. "On Axis-Parallel Tests for Tensor Product Codes". In: *Proceedings of the 21st International Workshop on Randomization and Computation*. RANDOM '17. 2017, 39:1–39:22.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-Quantum and Transparent Recursive Proofs from Holography". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 769–793.

[DI14] Erez Druk and Yuval Ishai. "Linear-time encodable codes meeting the Gilbert–Varshamov bound and their cryptographic applications". In: *Proceedings of the 5th Innovations in Theoretical Computer Science Conference*. ITCS '14. 2014, pp. 169–182.

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

[GI01] Venkatesan Guruswami and Piotr Indyk. "Expander-based constructions of efficiently decodable codes". In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. FOCS '01. 2001, pp. 658–667.

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

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

[GW20] Ariel Gabizon and Zachary J. Williamson. "plookup: A simplified polynomial protocol for lookup tables". In: (2020).

[Gro09] Jens Groth. "Linear Algebra with Sub-linear Zero-Knowledge Arguments". In: *Proceedings of the 29th Annual International Cryptology Conference*. CRYPTO '09. 2009, pp. 192–208.

[IW14] Yuval Ishai and Mor Weiss. "Probabilistically Checkable Proofs of Proximity with Zero-Knowledge". In: *Proceedings of the 11th Theory of Cryptography Conference*. TCC '14. 2014, pp. 121–145.

[KKB88] Michael Kaminski, David Kirkpatrick, and Nader Bshouty. "Addition Requirements for Matrix and Transposed Matrix Products". In: *Journal of Algorithms* 9.3 (1988), pp. 354–364.

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

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

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

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

[OWWB20] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. “Scaling Verifiable Computation Using Efficient Set Accumulators”. In: *Proceedings of the 29th USENIX Security Symposium*. Security ’20. 2020, pp. 2075–2092.

[Pip80] Nicholas Pippenger. “On the Evaluation of Powers and Monomials”. In: *SIAM Journal on Computing* 9.2 (1980), pp. 230–250.

[RR20] Noga Ron-Zewi and Ron Rothblum. “Local Proofs Approaching the Witness Length”. In: *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*. FOCS ’20. 2020.

[RRR16] Omer Reingold, Ron Rothblum, and Guy Rothblum. “Constant-Round Interactive Proofs for Delegating Computation”. In: *Proceedings of the 48th ACM Symposium on the Theory of Computing*. STOC ’16. 2016, pp. 49–62.

[Set20] Srinath Setty. “Spartan: Efficient and general-purpose zkSNARKs without trusted setup”. In: *Proceedings of the 40th Annual International Cryptology Conference*. CRYPTO ’20. Referencing Cryptology ePrint Archive, Report 2019/550, revision from 2020.02.28. 2020.

[Spi96] Daniel A. Spielman. “Linear-time encodable and decodable error-correcting codes”. In: *IEEE Transactions on Information Theory* 42.6 (1996). Preliminary version appeared in STOC ’95., pp. 1723–1731.

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

[Vid15] Michael Viderman. “A combination of testability and decodability by tensor products”. In: *Random Structures and Algorithms* 46.3 (2015). Preliminary version appeared in APPROX-RANDOM ’12., pp. 572–598.

[WTSTW18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. “Doubly-efficient zkSNARKs without trusted setup”. In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P ’18. 2018, pp. 926–943.

[XZZPS19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO ’19. 2019, pp. 733–764.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. “Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof”. In: *Proceedings of the 41st IEEE Symposium on Security and Privacy*. S&P ’20. 2020, pp. 859–876.