Linear-Time Arguments with Sublinear Verification from Tensor Codes

Jonathan Bootle[1](✉), Alessandro Chiesa[1], and Jens Groth[2]

[1] UC Berkeley, Berkeley, USA
{jonathan.bootle,alexch}@berkeley.edu
[2] Dfinity, Zürich, Switzerland
jens@dfinity.org

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

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

© International Association for Cryptologic Research 2020
R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12551, pp. 19–46, 2020.
https://doi.org/10.1007/978-3-030-64378-2_2

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 Fig. 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 $\mathsf{tp} = O(N)$, query complexity $\mathsf{q} = O(\sqrt{N})$, and verifier time $\mathsf{tv} = O(\sqrt{N})$. (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 $O(\mathsf{q} \cdot \log N)$. In this latter step, if one can evaluate the hash function in linear time, the resulting argument prover runs in time $O(\mathsf{tp})$ and $O(\mathsf{tv})$. 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 Fig. 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 $O(1/\epsilon + \log M)$;*

[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.

- *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 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 given in the full version of this paper .

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 Sect. 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 Sect. 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 the full version of this paper. 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 Sect. 2.2 and 2.3. The technical details are in the full version of this paper. 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

	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	$O_\lambda(N^\epsilon)$	✓

Fig. 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	$O(N^\epsilon)$

Fig. 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).

test for the Reed–Solomon code in [BBHR18]; and the code-switching technique in [RR20] for systematic linear codes.

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 Sect. 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 Sect. 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 Sect. 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

[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.

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.

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 $(q_0, q_1, \dots, q_t)$ and then receives as answer the linear combination $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_i \rangle$, where $q_1, \dots, q_t \in \mathbb{F}^k$ 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 the full version of this paper. 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.

[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.).

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 given in the full version.) 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 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 $\mathit{\Pi} \in \mathbb{F}^{k^t}$ that should be sent to $\mathbf{V}$, $\hat{\mathbf{P}}$ sends to $\hat{\mathbf{V}}$ an encoded proof string $\hat{\mathit{\Pi}} := \mathrm{Enc}(\mathit{\Pi}) \in \mathbb{F}^{n^t}$, 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 $\mathit{\Pi}_i$, $\hat{\mathbf{V}}$ forwards this query (as a message) to $\hat{\mathbf{P}}$, who replies with a "short" proof message that contains the answer $\langle q_1 \otimes \cdots \otimes q_t, \mathit{\Pi}_i \rangle \in \mathbb{F}$; 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 $\hat{\mathit{\Pi}}_1, \ldots, \hat{\mathit{\Pi}}_\ell \in \mathbb{F}^{n^t}$ 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 $\mathit{\Pi}_1, \ldots, \mathit{\Pi}_\ell \in \mathbb{F}^{k^t}$. 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 $q = (i, q_1, \dots, q_t)$ then $v_q = \langle q_1 \otimes \dots \otimes q_t, \Pi_i \rangle$.

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 $\hat{\Pi}_1, \dots, \hat{\Pi}_\ell$ 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 $q_0 \otimes q_1 \otimes \dots \otimes q_t \in \mathbb{F}^{\ell \cdot k^t}$ to the "stacking" of all proof strings. Namely, $\hat{\mathbf{P}}$ claims that the stacked word $\hat{\Pi} := \mathrm{Stack}(\hat{\Pi}_1, \dots, \hat{\Pi}_\ell) \in \mathbb{F}^{\ell \cdot n^t}$ can be decoded to some stacked proof string $\Pi := \mathrm{Stack}(\Pi_1, \dots, \Pi_\ell) \in \mathbb{F}^{\ell \cdot k^t}$ such that $v = \langle q_0 \otimes q_1 \otimes \dots \otimes q_t, \Pi \rangle$.[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 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, $c_1, \dots, c_t$ will help $\hat{\mathbf{V}}$ perform the consistency check that the value $v \in \mathbb{F}$ is the answer to the tensor query $q_0 \otimes q_1 \otimes \dots \otimes q_t \in \mathbb{F}^{\ell \cdot k^t}$.

- 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

[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 $\langle q_0 \otimes q_1 \otimes \dots \otimes q_t, \Pi \rangle = \langle q_1 \otimes \dots \otimes q_t, \Pi_i \rangle$.

the values of $\mathrm{Fold}(\hat{\Pi}; q_0): [n]^t \to \mathbb{F}$ 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:

[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) $O(d^t/|\mathbb{F}|) + (1 - \Delta)^\lambda$.

– 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 $\ell \cdot n^t = \frac{n^t}{k^t} \cdot \ell \cdot k^t = \rho^{-t} \cdot \mathsf{l}$. (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 Sect. 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 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 Sect. 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 the full version of this paper. 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, $\text{Fold}(\,\cdot\,; \alpha)$ 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.

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.

[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 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. Fold $(c_{r-1}, d_{r-1}; \alpha_r, \beta_r) = \mathrm{enc}(d_r)$ 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 Sect. 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 $\gamma_A = \langle r_A, z\rangle, \gamma_B = \langle r_B, z\rangle, \gamma_C = \langle r_C, z\rangle$.

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 subroutine 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 the full version of this paper 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 $u = (u_1, \dots, u_N)$ and $v = (v_1, \dots, v_N)$ in $\mathbb{F}^N$ with respect to a third vector $y = (y_1, \dots, y_N)$ 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:*

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

- *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 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 the full version of this paper. 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 Sect. 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 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 $U(X_1, \dots, X_l)$ and $V(X_1, \dots, X_l)$ over $\mathbb{F}$. Namely, in $U(X)$, we replace the monomial X^i, which has coefficient u_i, with a monomial in formal variables $X_1, X_2, \dots, X_{\log N}$, 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 $X_1, \dots, X_{\log N}$. 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 $W(X_1, \dots, X_{\log N}) := U(X_1, \dots, X_{\log N}) \cdot V(X_1, \dots, X_{\log N})$ 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 $p(X_1, \dots, X_l)$ a polynomial over $\mathbb{F}$. Then for every integer vector $\vec{j} = (j_1, \dots, j_l) \in \mathbb{N}^l$,*

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

Above we denote by $p_{\vec{\imath}}$ 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 $\vec{j} := (1, \dots, 1)$. 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 $\rho_1, \dots, \rho_l$. 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 the full version of this paper.

Remark 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. 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 Sect. 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 Sect. 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 $\langle r^{\intercal} U, v \rangle = r^{\intercal} U v$, and enlists the prover's help to do so. Therefore, the verifier sends $r_1, \ldots, r_t$ to the prover, who replies with $\gamma := r^{\intercal} U v \in \mathbb{F}$. 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 $r^{\intercal} U v$ in terms of the sparse representation of U:

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

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 Eq. (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 Sect. 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 the full version of this paper, 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 the full version.

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 1 ([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*

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

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 $w_{\circlearrowright} := \mathrm{shift}(w)$ and $b_{\circlearrowright} := \mathrm{shift}(b)$. 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 Eq. (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; the full version of this paper for details.

[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 Eq. (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.
- 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.

The details of both subprotocols can be found in the full version of this paper.

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] Applebaum, B., Haramaty-Krasne, N., Ishai, Y., Kushilevitz, E., Vaikuntanathan, V.: Low-complexity cryptographic hash functions. In: Proceedings of the 8th Innovations in Theoretical Computer Science Conference, ITCS 2017, pp. 1–31, (2017)

[AHIV17] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 24th ACM Conference on Computer and Communications Security, CCS 2017, pp. 2087–2104 (2017)

[BBBPWM18] Benedikt, B., Bootle, J., Dan, B., Andrew, P., Pieter, W., Greg, M.: Bulletproofs: short proofs for confidential transactions and more. In: Proceedings of the 39th IEEE Symposium on Security and Privacy, S&P 2018, pp. 315–334 (2018)

[BBCGI19] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 67–97. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_3

[BBHR18] Ben-Sasson, E., Bentov, I., Horesh, Y. and Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. In: Proceedings of the 45th International Colloquium on Automata, Languages and Programming, ICALP 2018. pp. 1–17 (2018)

[BCCGP16] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12

[BCGGHJ17] Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 336–365. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_12

[BCGJM18] Bootle, J., Cerulli, A., Groth, J., Jakobsen, S., Maller, M.: Arya: nearly linear-time zero-knowledge proofs for correct program execution. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11272, pp. 595–626. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03326-2_20

[BCRSVW19] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4

[BCS16] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2

[BFS20] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_24

[BS06] Ben-Sasson, E.: Sudan, madhu: robust locally testable codes and products of codes. Random Struct. Alg. **28**(4), 387–402 (2006)

[Ben+14] Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy, SP 2014. pp. 459–474, (2014)

[CHMMVW20] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26

[CMS17] Chiesa, A., Manohar, P., Shinkar, I.: On axis-parallel tests for tensor product codes. In: Proceedings of the 21st International Workshop on Randomization and Computation, RANDOM 2017. pp. 1–22, (2017)

[COS20] Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_27

[DI14] Druk, E., Ishai, Y.: 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 2014. pp. 169–182 (2014)

[GKR08] Shafi, G., Yael, T.K., Guy, N.R.: Delegating computation: interactive proofs for Muggles. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008. pp. 113–122 (2008)

[GW20] Ariel, G., Zachary J.W.: plookup: A simplified polynomial protocol for lookup tables (2020)

[Gro09] Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 192–208. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_12

[KKB88] Kaminski, M.: Kirkpatrick, David, Bshouty, Nader: addition requirements for matrix and transposed matrix products. J. Alg. **9**(3), 354–364 (1988)

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

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

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

[OWWB20] Ozdemir, A., Wahby, R., Whitehat, B., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: Proceedings of the 29th USENIX Security Symposium, Security 2020. pp. 2075–2092 (2020)

[Pip80] Pippenger, N.: On the evaluation of powers and monomials. SIAM J. Comput. **9**(2), 230–250 (1980)

[RR20] Ron-Zewi, N., Rothblum, R.: Local proofs approaching the witness length. In: Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science FOCS 2020 (2020)

[RRR16] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Proceedings of the 48th ACM Symposium on the Theory of Computing, STOC 2016. pp. 49–62 (2016)

[Set20] Setty, S.: Spartan: efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_25

[Tha13] Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 71–89. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_5

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

[WTSTW18] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J, Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: Proceedings of the 39th IEEE Symposium on Security and Privacy, S&P 2018, pp. 926–943 (2018)

[XZZPS19] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_24

[ZXZS20] Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: Proceedings of the 41st IEEE Symposium on Security and Privacy, S&P 2020. pp. 859–876 (2020)